НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Модуль підготовки односторінкових застосунків до публікації»

Виконав:
студент IV курсу, групи ІО-63
Панасюк Олександр Миколайович _____

Керівник:
Доцент кафедри ОТ, к.т.н.,
Болдак Андрій Олександрович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент кафедри АСОІУ, к.т.н.,
Ліщук Катерина Ігорівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп’ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Панасюк Олександр Миколайович

1. Тема проєкту «Модуль підготовки односторінкових застосунків до публікації», керівник проєкту Болдак Андрій Олександрович, доцент, к.т.н, затверджені наказом по університету від 07 травня 2020р. №1081-с

2. Термін подання студентом проєкту 20 травня 2020р.

3. Вихідні дані до проєкту технічне завдання, теоретичні данні

4. Зміст пояснювальної записки опис предметної області, дослідження систем збірки односторінкових застосунків, опис програмного продукту

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

Структурна схема системи, принципова схема, блок-схема алгоритму.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормо контроль	Сімоненко В.П. доктор технічних наук		

7. Дата видачі завдання 20.02.2020 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми роботи	15.11.2019	
2.	Вивчення та аналіз завдання	30.03.2020	
3.	Розробка архітектури та загальної структури систем	10.04.2020	
4.	Розробка структур окремих підсистем	22.04.2020	
5.	Програмна реалізація системи	28.04.2020	
6.	Оформлення пояснювальної записки	04.05.2020	
7.	Захист програмного продукту	14.05.2020	
8.	Передзахист	26.05.2020	
9.	Захист	20.06.2020	

Студент Олександр ПАНАСЮК

Керівник Андрій БОЛДАК

Анотація

Пояснювальна записка дипломного проєкту складається з трьох розділів, містить 5 таблиць, 3 додатки та 15 джерел – загалом 67 сторінки.

Об’єкт дослідження: Модуль підготовки односторінкових застосунків до публікації.

Мета дипломного проєкту: розширення інструментарію розробників з допомогою розробки спеціального синтаксису та модуля підготовки односторінкових застосунків до публікації, що дозволить зменшити час необхідний для завантаження та обробки скриптів.

У першому розділі було розглянуто загальні відомості, проблематику та історію розвитку модулів підготовки. Проаналізовано актуальність та необхідність розробки модуля.

У другому розділі описано предметну область та визначено функціональні вимоги до модуля. Розроблено алгоритми роботи модуля підготовки односторіноквих застосунків для публікації, враховуючи обрані тезнології

У третьому розроблено програмний додаток. Проведено тестування програмного забезпечення методом автоматичного модульного тестування. Проаналізовано роботу модуля на тестовому проєкті.

Ключові слова: Модуль підготовки, завантажувач, односторінковий застосунок

Annotation

The explanatory note of the diploma project consists of three sections, contains 5 tables, 3 annexes and 15 sources - a total of 67 pages.

The object of study: Preparation module for single-page application publication.

The aim of the diploma project: to expand the tools of developers by developing a special syntax and module for preparing single-page applications for publication, which will reduce the time required to download and process scripts.

In the first section the general information, problems and history of development of preparation modules were considered. The relevance and necessity of module development are analyzed.

The second section describes the subject area and defines the functional requirements for the module. Algorithms of work of the module of preparation of one-page applications for the publication are developed, considering the chosen technologies.

In the third the software application is developed. The software was tested by the method of automatic modular testing. The work of the module on the test project is analyzed.

Keywords: Preparation module, loader, single-page application

Опис альбому
дипломного проєкту

на тему: «Модуль підготовки односторінкових застосунків до публікації»

Київ – 2020

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6323 00.000 ВП	Відомість дипломного проєкту	1	
3	А4	ДП 6323 01.000 ТЗ	Технічне завдання	3	
4	А4	ДП 6323 02.000 ПЗ	Пояснювальна записка	67	
5	А3	ДП 6323 03.000 Д1	Клієнт серверна взаємодія із проектом Схема структурна	1	
6	А3	ДП 6323 04.000 Д2	Взаємодія та передача даних між компонентами Webpack Схема функціональна	1	
7	А3	ДП 6323 05.000 Д3	Алгоритм роботи завантажувача Схема принципова	1	

				ДП 6323 00.000 ВП		
	ПІБ	Підп.	Дата			
Розробн.	Панасюк О.М.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Болдак А.О.					
					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-63	
Н/Контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

Технічне завдання
до дипломного проєкту

на тему: «Модуль підготовки односторінкових застосунків до публікації»

Київ – 2020

ЗМІСТ

				ДП 6323 01.000 ТЗ		
	ПІБ	Підп.	Дата			
Розробн.	Панасюк О.М.			Технічне завдання	Лист	Листів
Керівн.	Болдак А.О.				1	3
					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-63	
Н/Контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку модуля підготовки односторінкових застосунків до публікації. Область застосування: розробка клієнтської частини веб-застосунків.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут» Ім. Ігоря Сікорського.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка модуля підготовки односторінкових застосунків до публікації та моделювання використання такої системи.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, а також публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Можливість конфігурації синтаксису інструкцій для модуля, щоб забезпечити гнучкість використання.
- Надання розробнику інструментів для контролю процесу збірки проекту, що надає змогу зменшити час необхідний для завантаження та обробки скриптів;
- Тестування розробленого додатку;
- Можливість запису логів під час збірки, для кращого розуміння процесу та швидшого пошуку помилок .

5.2. Вимоги до програмного забезпечення

- Операційна система Linux, macOS, Windows 7, Windows Server 2008 R2 або новіше;
- Node.js 8 або новіше;
- Npm 5 або новіше;

5.3. Вимоги до апаратної частини

- Комп’ютер на базі AMD Athlon 200GE або новіше;
- Оперативної пам’яті не менше 2 Гбайт;
- Вільний простір жорсткого диску не менше 200 Мбайт;
- Підключення до інтернету;

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	22.03.2020
Складання і узгодження технічного завдання	24.03.2020
Аналіз структури програмного забезпечення	03.04.2020
Створення модулів системи, що розробляється	25.04.2020
Тестування окремих модулів системи	01.05.2020
Доопрацювання, налагодження і виправлення помилок	21.05.2020
Оформлення документації дипломної роботи	22.05.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Модуль підготовки односторінкових застосунків до публікації»

Київ – 2020 року

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

SAP – single-page application – Односторінковий додаток

JS – JavaScript – мова програмування JavaScript

MPA – multi-page application – Багатосторінковий додаток

API – application programming interface – Прикладний програмний інтерфейс

CLI – Command Line Interface – Інтерфейс командного рядка

NPM – Node Package Manager – Менеджер пакунів

IDE – Integrated Development Environment – Інтегроване середовище розробки

JSON - JavaScript Object Notation - текстовий формат обміну даними, похідний від об’єкту мови програмування JavaScript

ВСТУП

Розвиток веб-застосунків завжди був нелінійний та непередбачуваний, оскільки середа виконання та інструменти мали постійний, але хаотичний розвиток. Звичайно це вплинуло на розвиток мов програмування та засобів розробки. Односторінкові застосунки відносно нове явище у розвитку та становленні веб-застосунків, однак не позбавлене усіх проблем минуло.

Сучасні SPA складаються із сотень або навіть тисяч компонентів. Код програми та вихідний код програми - це абсолютно різні речі. Під підготовки до публікації відбувається величезна кількість змін та оптимізацій програми. Але цей потужний інструмент можливо використовувати більш ефективно, якщо надати розробниками можливість безпосередньо у програмі залишати інструкції для таск-менеджера.

Існують подібні модулі для певних таск-менеджерів, проте вони мають низку проблем, які вимагають дослідницького рішення. Функціонал цих модулів є примітивним, а універсальними їх назвати взагалі не є можливим.

До прикладу візьмемо модуль webpack-conditional-loader для системи збірки webpack. Жодних конфігурацій дане рішення не має. Розробнику надається єдиний можливий інтерфейс та формат інструкції, тобто умова початком умовного блоку є // #if а кінцем // #endif. Тобто дане рішення абсолютно не пристосоване для мов у яких коментарі починаються не з “//”.

Основною метою розробки модулю підготовки є зменшення розміру кінцевої програми. При цьому модуль повинен бути адаптивний і працювати не лише з файлами мови програмування, а й розмітки сторінок та файлами стилів, для покращення та потенційно більших можливостей зменшення розміру програми.

Відповідно до статистики, сайту httparchive.org розміри веб-сторінок постійно збільшуються. За період з 11.05.2010 по 01.01.2020 розміри змінились наступним чином:

1. Загальний середній розмір веб-сторінок збільшився на 321.1% з 467.7 кб до 1969.4 кб для настільних ПК та на 1122.8% з 144.8 кб до 1770.6 кб для мобільних платформ.

2. Середній розмір таблиць стилів для веб-сторінок збільшився на 297.5% з 16.3 кб до 64.8 кб для настільних ПК та на 398.3% з 12.0 кб до 59.8 кб для мобільних платформ.

3. Середній розмір скриптів для веб-сторінок збільшився на 381.6% з 88.6 кб до 426.7 кб для настільних ПК та на 649.6% з 52.4 кб до 392.8 кб для мобільних платформ.

З цієї статистики очевидно, що розміри веб-застосунків постійно зростають. Звісно, пропускна здатність інтернет мережі та швидкодія пристроїв за цей період часу теж зросла, однак це не вирішило проблема в цілому.

Величезні компанії такі як Amazon, Google, Walmart підрахували збитки, що вони і не тільки вони несуть через затримки в завантаженні веб-сторінок. І ось що їхні дослідження кажуть:

1. “Одна секунда затримки завантаження коштуватиме Amazon 1.6 мільярдів доларів у продажах за рік.” - Amazon.

2. “Якщо час завантаження збільшиться з 1 секунди до 4 секунд, рівень конверсії значно знизиться. За кожну секунду пришвидшення завантаження, ми очікуємо приріст на конверсії на 2%.” - Walmart.

3. “Затримка відображення пошукових запитів у 400 мс призведе до зниження трафіку на 0.44% . В масштабах компанії - це 440 мільйонів перерваних сесій на місяц і величезні втрати з рекламних оголошень.” - Google.

Саме тому я хочу розробити власний модуль підготовки односторінкових застосунків для публікації, адже ця тема зараз є актуальною як ніколи і має потенціал покращувати швидкодію застосунків та зменшувати втрати бізнесу за рахунок зменшення часу завантаження.

РОЗДІЛ 1. ОГЛЯД ТЕХНОЛОГІЙ ТА ГОТОВИХ РІШЕНЬ

1.1 Засоби розробки модулів підготовки односторінкових застосунків до публікації.

Модулі підготовки односторінкових застосунків до публікації є відносно новим підходом у веб-розробці. Веб-застосунки пройшли чималий шлях і кардинально змінились за останні 20 років. Від найпростіших сторінок, що відображали текстову інформації та мали посилання на інші сторінки, до сайтів з продажу із моделювання предметів у 3D та складних банківських застосунків.

Сучасну людина дивиться фільми, купує речі, отримує зарплату через веб-застосунки та навіть не замислюється про це. Рівень складності та відповідальності програм клієнтської сторони виріс настільки, що звичних інструментів розробки веб-застосунків стало недостатньо.

Функціонал модулів підготовки визначають потреби веб-застосунків.

На сьогоднішній день модулі підготовки виконують величезну кількість різних операцій. Більшість проблем, що вони вирішують так чи інакше пов’язані із середовищем виконання веб-застосунків та його обмеженнями.

Потрібно виділити дві основні проблеми, що вирішують модулі підготовки.

Першою є швидкість завантаження. Розробники постійно намагаються зменшити об’єм своїх програм, щоб користувач якомога швидше міг завантажити сторінку. Здається, що це не так важливо, однак статистика великих компаній говорить про інше.

Друга проблема, та не менш важлива, підтримка нових функцій застарілими браузерами. Постійний розвиток веб-застосунків надає розробниками все нові та нові інструменти, але чимало користувачів користуються застарілими браузерами без підтримки нового функціоналу. Тому розробникам потрібно забезпечити стабільну роботу їх застосунків, навіть в збиток швидкодії або загальному розміру програми.

Одними з перших з’явились модулі підготовки що мінімізували html, css та js сирцевий код застосунків. Компілятору, на відміну від людини, байдуже чи відповідає код правилам форматування та чи зручно його читати, головне це правильний синтаксис. Тому були створені невеликі модулі, що вирізали зайві відступи та коментарі. Це зменшувало загальний розмір веб-сторінки та пришвидшило завантаження сайтів. Цей метод використовується до сих пір і є базовим для будь-якого веб-застосунку, що претендує на високі місця у пошукових запитах.

Наступним етапом були модулі компіляції таблиць стилів. Вихід CSS3 значною мірою йцу можливості стилізації веб-сторінок, однак не всі браузери були готові до цього і велика кількість правил потребувала вендорні префікси. Також почали з’являтись надбудови над CSS такі як: LESS, SASS, Stylus та інші. Вони потребували механізм перетворення їхнього синтаксису в зрозумілий браузеру CSS. Так на світ з’явились пре-процесори та пост-процесори таблиць стилів.

Ще одним суттєвим етапом став перехід на складних веб-застосунків на модульну системи і потрібність менеджера пакетів для проектів. Сучасні веб-застосунки мають тисячі компонентів та десятки або навіть сотні залежностей. І ось, ми вже впритул наблизились до інструментів без яких неможливе створення SPA. Саме в цей період набули популярності та розвинулись пакетні менеджери Bower і Npm та таск менеджери Gulp, Webpack та інші, але про них трошки пізніше.

Одним з ключових інструментів, що значно пришвидшили розвиток, модулів підготовки застосунків до публікації стала платформа NodeJs. Платформа надавала водночас як і прості та зручні інструменти для роботи із файлами, базами даних та виконанням скриптів, так і широкі функціональні можливості не поступаючись іншим серверним мовам програмування. При цьому кожен розробник отримав можливість конфігурувати та вдосконалювати модулі, адже вони написані мовою програмування JavaScript, що добре відома всім веб-розробникам.

Очевидно, що для створення модулю підготовки односторінкового застосунки для публікації потрібно чітко розуміти проблеми SPA та розуміти яку з них потрібно вирішити. Розглянемо детальніше, що ж таке односторінкові застосунки.

1.2 Односторінкові застосунки (SPA)

Як зрозуміло із назви, SPA[1] - це веб-сайт або застосунок, який містить весь його вміст на одній сторінці[2]. Натискання певних кнопок, закладок і посилань не призводить до перезавантаження сторінки. Завантажуються лише нові дані та змінюється стан сторінки, що призводить до оновлення її змісту.

На відміну від цього, багатосторінкові веб-сайти зазвичай завантажують абсолютно нові сторінки у відповідь на дії користувача. В результаті користувач змушений чекати завантаження сторінки, навіть якщо більша частина вмісту сторінки залишається такою ж.

Концепція не нова, але все ж привертає багато уваги і з’являється у заголовках дискусій. Світові гіганти, такі як Google і Facebook[3], впровадили це рішення у кількох своїх продуктах, тому ігнорувати його чи недооцінювати його неможливо.

Що ж таке односторінковий застосунок? Односторінковий застосунок або SPA - це веб-рішення, яке відображає JS-код безпосередньо у браузері та не вимагає перезавантаження сторінок під час роботи із користувачем. Його початкове призначення - оптимізація мобільного досвіду; SPA створює зручний інтерфейс прямо у мобільному браузері користувача. Після першого відкриття сторінки всі дані завантажуються автоматично.

У MPA[4] користувачі переключаються між окремими сторінками, щоб отримати доступ до потрібного вмісту. Але в односторінкових застосунках користувачі переходять між різними станами сторінки. Натиснувши на посилання чи кнопку в SPA, сторінка не буде перезавантажена. У браузері відображається лише певний стан сторінки, і при запиті змінюється лише вміст сторінки, решта залишається незмінною.

Ця функція можлива завдяки AJAX технології, що дозволяє клієнту спілкуватися з сервером без перезавантаження сторінки.

Якщо говорити про односторінкові застосунки , ми буквально маємо на увазі, що у додатку є лише одна сторінка. Ця єдина HTML-сторінка забезпечує динамічні оновлення та значно зменшує навантаження на сервер. В результаті власник веб-сайту отримує прибуток у трьох різних сферах:

- швидкість завантаження - весь вміст завантажується автоматично на початку, тому користувачеві не доведеться чекати після натискання кожної кнопки на веб-сайті
- простота - SPA працює лінійно і не переривається перешкодами, такими як поганий зв'язок або застаріле програмне забезпечення
- Досвід користувачів - користувачі очікують, що ваш веб-сайт реагуватиме на їх запити не більше ніж 2 секунди, і якщо ваш веб-сайт швидко відповідає, ваш застосунок відповідає їхнім вимогам, що неодмінно впливає на популярність та зручність використання застосунку.

MPA або багатосторінковий додаток або типовий веб-сайт, до якого ми звикли, після кожної дії на веб-сайті запитує нову HTML-сторінку з сервера. Це призводить до численних запитів для обробки даних та затримок між клієнтською та серверною сторонами. Тепер давайте подивимось, чим відрізняється SPA. Як ми виявили раніше, SPA використовує технологію AJAX. Це набір методів, які дозволяють оновити лише частину програми, а не весь веб-додаток. SPA надсилає на сервер запит AJAX, повертає дані JSON або іншого формату, в залежності від налаштування серверної частини, з сервера та передає певні частини веб-сторінки безпосередньо у браузері.

Щоб остаточно зрозуміти різницю між SPA та MPA. Наведемо порівняльну таблицю за основними критеріями.

Таблиця 1.1

Порівняння SPA та MPA

	SPA	MPA
Швидкість	Початкове завантаження займає більше часу, ніж у випадку з MPA, але подальше використання програми проходить безшовно та швидше	Завантажується повільніше і перезавантажується щоразу, коли користувач натискає будь-яку кнопку
Безпека	Вразливі до нападів хакерів, безпека може бути покращена правильним підходом під час розробки	Потрібен ретельний підхід для захисту кожної окремої сторінки веб-сайту
Процес розробки	Можливо повторно використовувати код серверної частини. Відокремлені серверна та клієнтська частини.	Численні залежності між серверною і клієнтською частинами
JavaScript залежності	Наявні, тому залежно від типу та версії веб-переглядача можуть виникнути проблеми при початковому завантаженні.	Ніяких JS залежностей
Навігація за посиланнями	Одне єдине посилання на весь додаток, проте є можливість посилатися на певний вміст	Окреме посилання на кожну сторінку MPA
Інтерфейс користувача	Гнучкий, безперебійний і мобільний	Краща інформаційна архітектура

Розглянемо переваги односторінкових застосунків більш детально. Важливо розробити зручний та приємний інтерфейс для користувача. Постійне перезавантаження сторінок дратує, нудне і трудомістке. Для неспокійних Інтернет-мандрівників, які можуть використовувати ваш застосунок, поганий інтерфейс веб-сайтів є очевидною причиною відмовитися від нього. Немає відвідувачів - немає клієнтів - немає доходу. Цей простий ланцюг наглядно демонструє важливість утримати відвідувачів на сторінці.

Внесок у інтерфейс користувача - найбільша перевага односторінкових застосунків. Як вже згадували вище, швидкість завантаження сторінки та продуктивність веб-додатків із поганим підключенням до Інтернету - це перше, на що необхідно звернути увагу. Пам'ятаючи, що затримка на 1 секунду може коштувати мільярди доларів, це серйозна причина для впровадження моделі SPA.

SPA - найкращий спосіб продемонструвати, як працює міжплатформенний підхід. Він добре виглядає і працює в будь-якій операційній системі, у будь-якому браузері. Цей факт вигідний як інженеру програмного забезпечення, так і типовому користувачеві. Розробники створюють єдину базу коду, яка працює в будь-якому браузері; користувачі можуть насолоджуватися адаптивними програмами на будь-якому пристрої.

Завдяки технологіям, що дозволяють створити чудовий SPA, налагодження в сучасних браузерах не завдає жодних проблем. Для розробники не стане проблема знайти на клієнтський чи серверній частині код працює не вірно. Інструменти для налагодження навіть покажуть місце, де щось пішло не так.

Але не потрібно забувати про компроміси на які потрібно йти. Хоча щойно описували позитивні речі, щодо часу завантаження сторінки. Однак ми не обговорювали швидкість початкового завантаження, яка може бути достатньо великою. Для завантаження всього веб-сайту потрібно багато часу, коли ви відкриваєте його вперше. Це може стати причиною для потенційного клієнта натиснути кнопку «Закрити» і більше не повернутися. Один із розробників менеджера задач Webpack Сиан Ларкин[5] рекомендує, щоб додаток був готовий до роботи після завантаження перших 200кб. Увесь функціонал, що не потрібен одразу, краще динамічно завантажити у процесі роботи із застосунком.

Недостатком також є погона оптимізація у пошукових системах Якщо ви зацікавлені у високому рейтингу веб-сайту, то вам слід продумати як вирішити цю проблему найсучаснішим із доступних способів.

Наведомо 3 основні речі, які викликають проблеми з SEO:

- Односторінковий застосунок розміщує весь вміст на одній сторінці; відсутність унікальних посилань негативно впливає на SEO та ускладнює оптимізацію веб-сайту для маркетингових цілей. Поряд із цим виникає ще одна проблема. Користувач не може повертатися назад і назад в SPA: кнопки браузера перейдуть на сторінку, завантажену раніше, але не до попереднього стану SPA. Отже, проблеми можуть виникнути, коли ви перевірите історію свого браузера, щоб знайти певну сторінку з цінною інформацією: ви бачите лише посилання на весь SPA і пошук елемента, потрібного вам, потребує певного часу. Як ми вже згадували вище, існує лише одне посилання, яке веде нас через весь SPA. Таким чином, обмін певним вмістом стає складним завданням. Ви можете додати посилання лише на односторінковий застосунок, а не на конкретний фрагмент. Існує спосіб уникнути цієї проблеми, використовуючи методи глибокого зв'язку, щоб привернути користувачів до конкретних фрагментів вмісту. Сучасні браузери надають програмний інтерфейс історій для вирішення цих проблем, головне щоб розробники користувались ним.
- Наступна проблема обмежене семантичне ядро. Важко упакувати односторінковий веб-сайт із великою кількістю ключових слів, але це важливо для SEO. Існує спосіб подолати цю проблему: забезпечити кращий вміст і зосередитись на метаописах, тегах та якорях, базуючись на вже відомому програмному інтерфейсі історій історій.
- Питання безпеки та захисту інформації стоїть наразі як ніколи. Компанії можуть збанкрутувати через втрату персональних даних користувачів. Як повністю заснована на JavaScript програма, рішення на одній сторінці дуже чутливі до атак на міжсайтові сценарії (XSS). XSS - це спосіб вставити шкідливі сценарії у форми

вашого веб-сайту. Забезпечивши захист серверної частини програми та застосувавши конкретні методи безпеки SPA, ваша команда розробників може запобігти атакам XSS.[5]

Підсумовуючи, все згадане вище, односторінкові застосунки мають чималу кількість переваг при правильній розробці, вони дозволяють збільшити привабливість сторінки для користувачів та покращити конверсії продажів, але тільки за умови, що й недоліками буде приділена достатня кількість уваги та часу.

1.3. Мова програмування JavaScript

JavaScript - часто скорочується як JS - мова програмування, яка відповідає специфікації ECMAScript. JavaScript - це високорівнева, багато парадигмальна мова програмування, що є основною і фактично єдиною мовою програмування клієнтської частини веб застосунків[6].

Перша версія JavaScript була випущена в грудні 1995 року. Мову було розроблено спеціально для браузерів, тому і можливості були значно менші ніж у вже розповсюджених мовах як C або Java. Але з плином часу, потреби веб-застосунків змінювались і так саом змінювалися стандарти мови. Сьогодні, JavaScript одна із найпопулярніших мов програмування.

JavaScript - об’єктно орієнтована прототипна мова програмування з динамічною типізацією[7]. Однак, часто мову програми використовують функціональну парадигму та подіє-орієнтований підхід. Це є наслідком того, що більшість подій у веб-застосунку відбуваються асинхронно та спрацьовують всередині функцій зворотного виклику. Об’єктно орієнтований підхід реалізується за допомогою прототипів, саме тому відсутнє поняття класу. В основі прототипного програмування використовується узагальнені об'єкти, які потім можуть бути клоновані та розширені. Наслідування полів та методів реалізується через посилання на прототип, що міститься у кожному об’єкті.

JavaScript є мовою із компіляцією на льоту, тобто перед початком виконання програми. На початку свого існування, мова виконувалась інтерпретатором рядок за рядком, але у середині 2000 років стало зрозуміло, що потрібно шукати більш швидкий підхід. Майже всі виробники браузерів почали розробку своїх середовищ виконання, щоб досягти максимальної ефективності. Сьогодні найпопулярнішим таким середовищем є V8, розроблений компанією Google. Перед виконанням скриптів, вони спочатку компілюються і лише потім починають виконання. На етапі компіляції, компілятор може вносити оптимізації на основі аналізу коду та підготовлює програму до виконання, що призводить покращення ефективності.

Широко розповсюджені JS подібні мови такі як: TypeScript, CoffeeScript та інші. Код програм написаних такими мовами, транслюється в JavaScript і лише тоді виконується. Причиною їх виникнення стала необхідність додати певні можливості до JS, яких не вистачає розробникам. Яскравим прикладом є статична типізація, що стала можливою у TypeScript. Однак, не потрібно забувати, що JS-подібні мови компілюються в JS, тому усі додаткові можливості існують лише під час компіляції. Під час виконання вони зникають, адже виконується саме JS код.

Мова дуже швидко розвивається і з’являються нові технології, що розширюють її можливості. Із виходом платформи NodeJS, стало можливим виконання программ не лише всередині браузера на клієнтській частині, а й на серверній. Тобто розробнику не потрібно знати декілька мов програмування, щоб створити повноцінний застосунок. Іншим рішенням, що внесло вклад у розвиток та популярність JavaScript, став вихід платформи Electron. Вона надає можливість створювати застосунки для будь-яких сучасних платформ, як мобільних так і настільних.

1.4. Платформа NodeJS

NodeJs - платформа для виконання JavaScript з відкритим кодом, головною метою якого є виконання JavaScript окремо від веб-браузера[8]. Із

появою NodeJs у розробників з'явилась можливість використовувати JavaScript для консольних утиліт та розробки серверної частини веб-застосунків. Саме Node.js усіляко розвиває парадигму "JavaScript всюди", що об'єднує розробників веб-додатків навколо однієї мови програмування, а не різних мов для скриптів на сервері та клієнті.

Node.js приніс подіє-орієнтований підхід програмування на веб-сервери, що дозволяє розвивати швидкі веб-сервери написані на JavaScript. Розробники можуть створювати масштабовані сервери без використання потоків, використовуючи спрощену модель програмування на основі подій, яка використовує функції зворотного виклику для сигналізації про виконання завдання. Node.js пов'язує простоту мови JavaScript з потужністю мережевого програмування Unix.

Node.js був побудований на Google V8 JavaScript[9], оскільки він був відкритим на основі ліцензії BSD. Веб-сервери у повній мірі та із наданням простого інтерфейсу працюють із базовими інтернет протоколами, таким як HTTP, DNS, TCP. JavaScript також був відомою мовою, завдяки чому Node.js був доступний спільноті веб-розробників та швидко набув популярності.

JavaScript є єдиною мовою, яку Node.js базово підтримує , але існує багато мов, що компілюються в JS. Як результат, програми Node.js можна написати на CoffeeScript, Dart, TypeScript, ClojureScript та інших.

Додаток Node.js запускається в одному процесі, не створюючи нової потоку для кожного запиту. Node.js надає набір асинхронних примітивів вводу та виводу у своїй стандартній бібліотеці, які запобігають блокуванню коду JavaScript і, як правило, бібліотеки в Node.js записуються з використанням не блокуючих парадигм, що робить блокування під час роботи винятком, а не нормою.

Коли Node.js повинен виконати операцію вводу / виводу, як-от читання з мережі, доступ до бази даних або файлової системи, замість блокування потоку і витрачання очікування циклів процесора, Node.js відновить операції, коли відповідь повернеться. Це дозволяє Node.js обробляти тисячі одночасних

з'єднань з одним сервером, не вводячи тягар управління паралельністю, що може бути складним і потенційно небезпечним джерелом помилок.

Розглянемо більш детально технічні особливості платформи, щоб краще зрозуміти її переваги та недоліки.

Програми на NodeJs виконуються в одному потоці, яким керую цикл подій із використання неблокуючої системи введення та виведення даних, що надає змогу платформі підтримувати тисячі одночасних з'єднань, не втрачаючи ресурси на зміну контексту потоку. Система загального використання одного потоку для всіх запитів, що із використанням шаблону спостерігача, створена для розробки сильно зв'язаних додатків, де усі функції, що виконють операції введення та виведення, повинні викликати функції зворотнього виклику. Для керування однопотоковим циклом подій NodeJs застосовує бібліотеку libuv, яка, в свою чергу, базується на пулі потоків фіксованого розміру, який обробляє виклики неблокуючих асинхронних операцій введення та виведення.

Пул потоків обробляє виконання паралельних завдань у Node.js. Основна функція потоку викликає завдання до загальної черги завдань, які потоки в пулі потоків тягнуть і виконують. Власне неблокуючі виклики системні функцій, таких як мережеві, переводяться на неблокуючі сокети на інші ядра, в той час як потенційно блокуючі системні функції, такі як читання та запис файлів, виконуються блокуючим способом у власних потоках. Коли потік у пулі потоків виконує завдання, він інформує головний потік, який, у свою чергу, розблокується та виконає зареєстрований зворотній виклик.

Недоліком цього одно потокового підходу є те, що Node.js не дозволяє вертикальне масштабування, збільшуючи кількість ядер CPU машини, на якій він працює, без використання додаткового модуля, такого як кластер, StrongLoop Process Manager або pm2. За необхідності, розробники мають можливість змінювати кількість потоків у пулі потоків бібліотеки libuv, . Операційна система сервера, ймовірно, розподіляє ці потоки по декількох ядрах. Інша проблема полягає в тому, що тривалі обчислення та інші завдання,

пов'язані з процесором, заморожують весь цикл подій до завершення.

Платформа використовує пакетний менеджер npm. Npm - це попередньо встановлений менеджер пакетів для платформи Node.js. Він встановлює програми Node.js з реєстру npm, організовуючи встановлення та управління сторонніми програмами Node.js. Пакети в реєстрі npm можуть варіюватися від простих бібліотек-помічників, таких як Lodash, до менеджерів задач, таких як Grunt.

Node.js реєструється в операційній системі, щоб ОС повідомляла про підключення та видала зворотній виклик. Під час виконання Node.js кожне з'єднання є невеликим підрозділом купи. Традиційно відносно важкі процеси ОС або потоки обробляли кожне з'єднання. Node.js використовує цикл подій для масштабованості замість процесів або потоків. На відміну від інших керованих подіями серверів, цикл подій Node.js не потребує явного виклику. Натомість визначаються зворотні виклики, і сервер автоматично вводить цикл подій в кінці визначення зворотного виклику. Node.js виходить з циклу подій, коли більше немає зворотних викликів.

Те що Node.js, розроблений одно поточним, не означає, що ви не можете скористатися кількома ядрами для покращення ефективності. Дочірні процеси можна породжувати за допомогою API та створювати форки процесів, і вони розроблені так, щоб між ними було легко спілкуватися. На цьому ж інтерфейсі побудований модуль кластера, який дозволяє обмінюватися сокетами між процесами, щоб забезпечити балансування навантаження та комунікацію між вашими ядрами.

1.5. Системи онлайн створення застосунків

Чимала кількість веб-застосунків сьогодні створюється онлайн. Такі системи набирають популярності та щороку збільшують аудиторію і можливості. Саме тому розглянемо більш детально вони працють.

Конструктор застосунків Wix. Одним із найпопулярніших сервісів створення застосунків онлайн є Wix.[1-1]. Здебільшого його використовують маленькі компанії для своїх застосунків або ж не корпоративні клієнти.

Головною особливістю Wix є те, що платформа надає можливість пройти усі етапи створення веб-застосунку, починаючи з дизайну і закінчуючи публікацією в мережі інтернет.

Для початку роботи не потрібно бути програмістом і навіть не потрібно знати мови програмування. Конструктор застосунків надає інтуїтивно зрозумілий інтерфейс побудови застосунків. На рисунку 1.1 зображено інтерфейс конструктора застосунків під час редагування сайту.

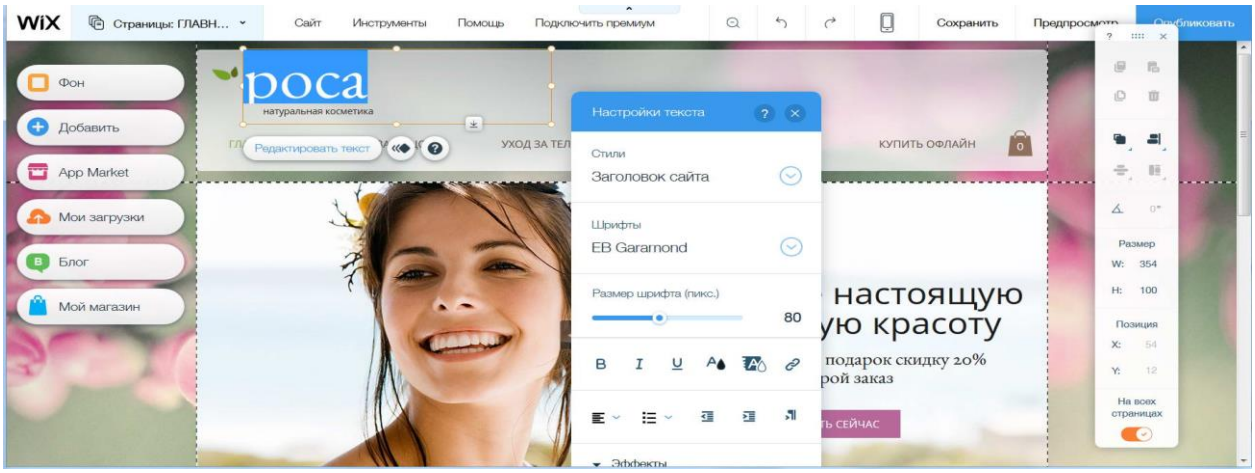


Рисунок 1.1. Інтерфейс конструктора застосунків Wix

Створення застосунку відбувається за наступним планом:

- Вибір шаблону. Платформа надає доступ до понад 500 різних шаблонів на вибір.
- Підключення та налаштування віджетів сайту.
- Заповнення контенту сайту.
- Підготовка до публікації. Вона включає в себе збереження веб-застосунку, налаштування домену та режиму доступу до нього.

Однак, платформа не надає можливості додавати свої віджети та не підтримує різні мови програмування. Також відсутня будь-яка можливість завантаження пакетів за допомогою менеджерів пакетів JavaScript. Ці

особливості унеможливлюють ручне вдосконалення сайту, що є досить неприємним і вагомим фактором для розробників веб-застосунків.

Конструктор Page Builder для WordPress. WordPress - це безкоштовна та відкрита система управління вмістом (CMS), написана на PHP та поєднана з базою даних MySQL або MariaDB. Особливості включають архітектуру

плагінів і систему шаблонів, що в WordPress називаються темами. Дана система управління використовується у чи малій кількості веб-застосунків, тому не дивно що і для неї було створено конструктори застосунків.

Одним з них є Page Builder[1-2]. Він надає змогу користувачеві, що не знає принципів розробки застосунків за допомогою інтерфейсу користувача створювати розмітку застосунку та налаштовувати віджети. На рисунку 1.2 зображено процес налаштування сайту у Page Builder.

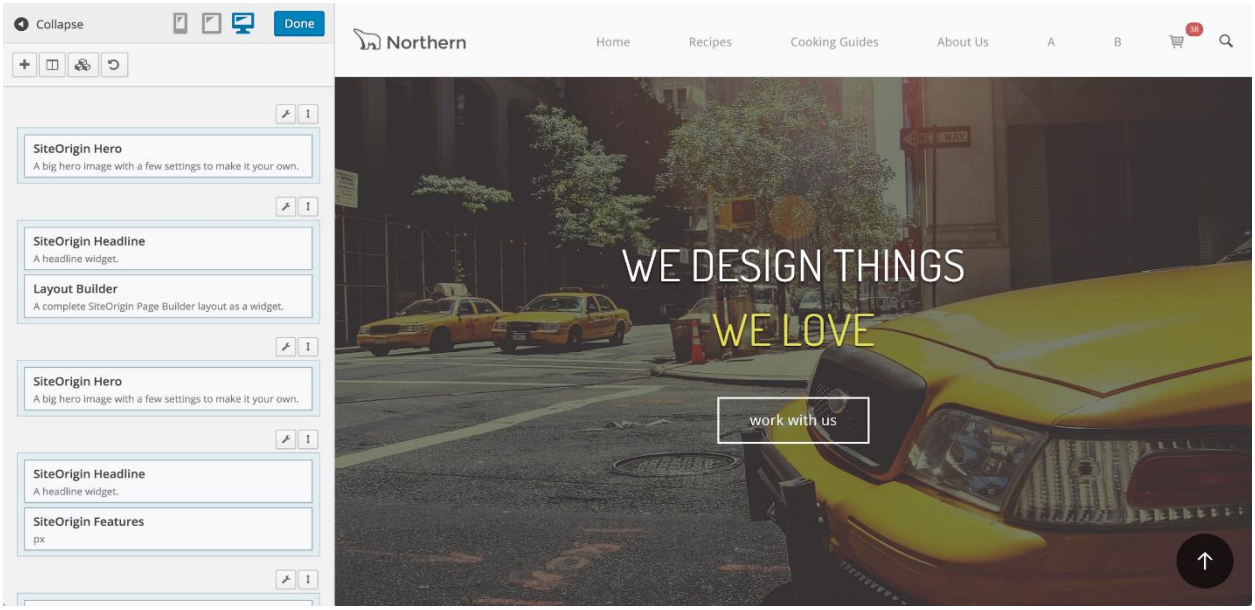


Рисунок 1.2. Процес налаштування сайту у Page Builder

Віджети завантажуються в комплекті із конструктором. Однак, можливості їх модифікації майже не існує, а додавати свої віджети не є можливим, що дуже сильно обмежує використання Page Builder, хоча для створення інтерфейсу користувача його можливості в купі із темами WordPress є достатньо не поганими.

Стримуючим фактором також є необхідність знання WordPress для можливості складних інтеграцій та не тривіальних завдань.

Онлайн платформа JACE - це платформа призначена для розробки SPA на основі використання віджетів.[10]

JACE можна позиціонувати як загальнодоступний онлайн-сервіс або як засіб швидкого прототипування застосувань для компаній, що займаються розробкою програмного забезпечення, ориентированного на кінцевого користувача.

JACE надає дружній користувальницький інтерфейс , заснований на drag & drop, використовує відкриту аутентифікацію Google, авторизацію доступу до додатків (адміністратор сервісу, автор і співавтор програми), надає розробнику можливість управління файлами на серверній стороні, має вбудовану i18n, використовує редактор Ace Editor в різних режимах (js, json, html, xml, csv, dot, dps) для редагування коду і налаштувань, дає можливість використовувати різноманітні шаблони сторінок і широку галерею віджетів. Відображає вид в режимах редагування і перегляду.

JACE надає більше 60 віджетів, починаючи з низькорівневих, наприклад таких як блок html, і закінчуючи такими, які реалізують мікро сервіси

Віджети рівня додатки дозволяють оформити статичну частину SPA такі як, банери, топбари, футери. Існує можлиість використовувати різноманітні інтерактивні графіки і схеми. Є віджети, які дозволяють створити і налагодити бізнес-логіку додатка, що виконується в сервісі обробки даних і в браузері. Спеціальна група віджетів дозволить Вам створювати опитування і управляти ними, а також отримувати у вигляді графіків результати статистичної обробки відповідей респондентів.

Для роботи із JACE необхідні навички програмування, адже платформе надає можливість власноруч створювати віджети та конфігурувати уже створені. На рисунку 1.3 зображена панель вибору віджета у JACE.

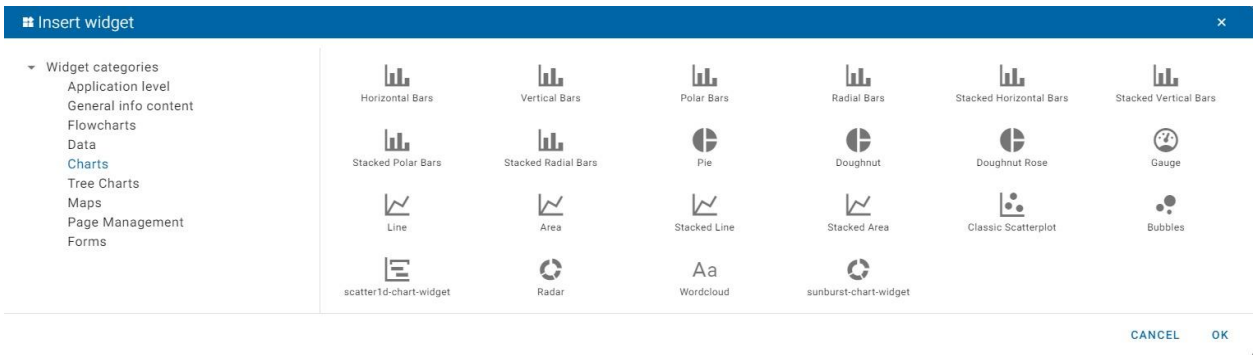


Рисунок 1.3. Вибір віджета у JACE

Порівняння систем онлайн створення застосунків. Кожна система має свої недоліки та переваги. Для того, щоб зрозуміти повноту картини необхідно порівняти їх.

Таблиця 1.2

Порівняння онлайн систем створення веб-застосунків

	Wix	Page Builder	JACE
Шаблони застосунків	+	+	-
Набір вбудованих віджетів	+	+	+
Створення власних віджетів	-	+	+
Необхідність вивчення додаткових технологій	-	+	+
Різні режими застосунку для розробника та користувача	-	+	-
Можливість покращення застосунку за межами системи	-	-	-
Використання менеджерів пакетів JS	-	-	-
Можливість видалення непотрібних модулів	-	-	-
Необхідність знань програмування	-	+	+

З порівняльної таблиці стає більш зрозуміло різницю між онлайн системами створення веб-застосунків. Кожна орієнтована на свого

користувача та надає певні переваги. Однак, дані системи абсолютно точно не є не універсальними.

Головним недоліком усіх представленим систем є те, що вони накладають певні обмеження на розробників. Не існує можливості напряму використовувати JavaScript пакети, що є у публічному доступі. Також, створені у них застосунки неможливо підтримувати та розвивати за межами платформ. Отже якщо вашому продукту не вистачає можливостей систему, необхідно створювати його з самого початку традиційним способом.

При цьому онлайн системи не підтримують явне виключення блоки програми розробником, що в купі з іншими недоліками роблять їх не найкращим вибором для створення професійних оптимізованих застосунків.

Враховуючи те, що модуль підготовки односторінкових веб-застосунків до публікації повинен бути дієвим та універсальним, його розробка не буде базуватись на цих системах.

ВИСНОВКИ ДО РОЗДІЛУ 1

В ході виконання даного розділу дипломного проєкту, було розглянуто проблематику веб-застосунків та їх підготовку до публікації. Представлені рішення та технології базуються на мові програмування JavaScript. Причина цього в тому, що ця мова стала основною у сучасному світі веб-застосунків та має дуже низький поріг входження. Але в процесі розгляду було з'ясовано, що дані рішення мають певні недоліки, які потрібно вирішити.

Основними недоліками є:

- Величезний розмір веб-застосунків, який з кожним роком зростає, при неможливості розробника напряму втручатись у цей процес.
- Існуючі рішення не є надто гнучкими і не забезпечують вирішення сучасних проблем. Тому розробнику вимушені створювати свої рішення, для кожного окремого випадку.
- Використання застарілих технологій, що зумовлює уповільнення розвитку веб-середовища в цілому.

Онлайн системи, на жаль, досі не є оптимізованими та гнучкими, саме через це я хочу розробити власний модуль підготовки односторінкових застосунків до публікації, що забезпечить вирішення описаних проблем. Він повинен бути максимально гнучким та конфігурованим заради зручності його використання та працювати, як для малих веб-застосунків, так і для складних односторінових застосунків, що налічують тисячі компонентів незалежно від технологій використаних у цих проектах.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОДУЛЯ ПІДГОТОВКИ ОДНОСТОРІНКОВИХ ЗАСТОСУНКІВ ДО ПУБЛІКАЦІЇ

В основі підходів розробки модуля підготовки односторінкових застосунків до публікації, лежать пакети та системи збірки застосунків, у які буде інтегровано модуль. Саме тому, потрібно детально розглянути існуючі пакетні менеджери, системи збирання модулів задач та їх можливості для вирішення проблем, що були розібрані у першому розділі.

2.1. Менеджери пакетів

Поява менеджерів пакеті повністю змінила щоденні процеси розробки на JavaScript, оскільки вони вирішили набір проблем та спростили використання мови. Перш за все, менеджери пакетів JavaScript впровадили новий надійний підхід до сторонніх програмних засобів. Головною ідеєю стало, що програміст повинен лише знати, як працює певний інструмент, а не вивчати алгоритм процесу інсталяції або як програмне забезпечення будується, компілюється та інтегрується в систему. Крім того, ви отримуєте рішення управління залежностями у проекті з кожним менеджером пакетів JavaScript. Програмне забезпечення знає залежності кожного пакету, тому автоматично встановлює всі необхідні компоненти, щоб задовольнити дерево залежностей, хоча до їх появи за це все відповідав розробник, що призводить до зменшення продуктивності та збільшення кількості помилок.

Крім того, кожен менеджер пакетів JS надає можливість оновлювати ваші залежності та пакети, тому ви завжди отримуєте найновіший функціонал або ви можете виконати оновлення вручну. Більше того, деякі менеджери пакунків надають можливість отримати нові функції або вирішити поточні проблеми за допомогою нового програмного забезпечення.

Якщо ви розробляєте пакети, ви також можете покластися на менеджерів пакетів, оскільки деякі з них надають інструменти для створення пакетів,

готових до розповсюдження. Вони дають вашим пакетам можливість побачити світ та бути корисними іншим робробникам. Створення такої екосистеми значним чином, вплинуло на розвиток веб-сервісів.

Менеджер пакетів Bower. Цей менеджер пакетів JavaScript спрощує щоденну роботу пов'язану із фреймворками, файлами, бібліотеками, утилітами та іншими компонентами веб-сайту. Bower допомагає керувати всіма цими речами. Він отримує та встановлює пакети, а також піклується про пошук, завантаження та збереження всіх необхідних матеріалів.

Bower використовує файл маніфесту bower.json для відстеження всіх ваших пакетів. Ви вирішуєте, як користуватися пакетами, але Bower пропонує деякі життєво важливі речі для полегшення цих процесів. Менеджер пакетів повністю оптимізований для розробки клієнтськой частини веб-застосунків та базується на плоскому дереві залежностей. Таким чином, Bower потребує лише однієї версії для кожного пакету і максимально скорочує час завантаження сторінки. Крім того, інструмент керує компонентами, що не належать JavaScript, та пропонує великий реєстр пакетів специфічного типу.

Менеджер пакетів Npm. Npm - менеджер пакетів JavaScript для Node.js. Він розпочався як проект з відкритим кодом, спрямований на допомогу розробникам JS для спільного використання пакетних модулів. Тепер, Npm Inc. - це компанія, яка також пропонує публічну колекцію пакетів Node.js, для веб-застосунків і мобільних додатків, маршрутизаторів, роботів та багатьох інших корисних речей, що зберігаються в реєстрі npm. Цей інструмент істотно спрощує такі рутинні процеси, як установка, пошук та публікація пакетів.

Npm складається з трьох різних компонентів:

- веб-сайт
- інтерфейс командного рядка (CLI)
- реєстр

Веб-сайт використовується , щоб відкрити пакунки, налаштувати профілі та керувати іншими аспектами npm . Наприклад, ви можете створити організацію для управління доступом до публічних або приватних пакетів.

Також, це зручний інструмент для пошуку потрібних саме вам пакетів. Веб-сайт надає можливість публікувати документацію до пакетів безпосередньо на ньому. Тому, нові користувачі одразу знають про можливості та способи використання пакетів.

CLI працює в терміналі, і це саме те місце, де більшість розробників взаємодіє з Npm. Не дивлячись на свою простоту, інтерфейс надає повний спектр можливостей пакетного менеджера, серед них: пошук пакетів, встановлення та деінсталяція пакетів і залежностей, управління версіями пакетів та запуск скриптів для полегшення рутинних завдань

Реєстр - це велика загальнодоступна база даних програмного забезпечення JavaScript та метаінформації, що його оточує. Кожен розробник використовує його через веб-сайт або програмний інтерфейс командного рядка. Це найбільш важлива частина пакетного менеджера, адже без неї інші не мають жодного сенсу.

Порівняння менеджерів пакетів. Оскільки менеджер пакетів, визначає шлях розповсюдження застосунку його вибір значним чином впливає на можливість використання модулів та їх розповсюдження. Заради наглядності наведемо порівняльну таблицю менеджерів пакетів за основними параметрами.

Таблиця 2.1

Порівняння менеджерів пакетів Npm та Bower

	Npm	Bower
Кількість пакетів	більше 200 000	36 000
Керування модулями	Присутнє	Присутнє
Частина NodeJS	Так	Ні
Вихід нових версій	Постійно оновлюється	Не оновлюється з 2017
Тип дерева залежностей	Вкладене	Плоске

2.2. Менеджери задач JavaScript

Перед початком аналізу існуючих менеджерів задач, потрібно чітко усвідомлювати їх можливості та чи відповідають вони вимогам проектного рішення. Тому, визначимо перелік можливостей, які повинні бути присутні у менеджері задач.

Основними вимогами є:

- сучасність - система повинна працювати із популярними фреймворками для розробки SPA;
- конфігурованість - система повинна надавати простий інтерфейс розробнику для конфігурування модулів;
- гнучкість - система повинна працювати не лише із файлами JavaScript, а й TypeScript, Html, Css та іншими.
- функціональність - повинна бути можливість реалізовувати будь-які операції із файлами та аналізувати сирцевий код застосунків, щоб не створювати бар'єрів для розробників.
- Визначивши основні вимоги, потрібно проаналізувати існуючі менеджери задач.

Grunt - це менеджер задач JavaScript, інструмент, який використовується для автоматичного виконання частих завдань, таких як мінімізація, компіляція, тестування та збірка застосунка. Він використовує інтерфейс командного рядка для запуску спеціальних завдань, визначених у файлі.

Менеджер задач виконується платформою NodeJs та розповсюджується через менеджер пакетів Npm. Перераховані технології активно використовуються більшістю веб-розробників, тому це однозначно є його перевагою.

Конфігурація Grunt задається в спеціально файлі конфігурацій gruntfile.js, Формат даних у цьому файлі конфігурації є JSON. На рисунку 2.1 зображено типовий файл конфігурації Grunt.

```
module.exports = function(grunt) {

  grunt.initConfig({
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
      options: {
        globals: {
          jQuery: true
        }
      }
    },
    watch: {
      files: ['<%= jshint.files %>'],
      tasks: ['jshint']
    }
  });

};
```

Рисунок 2.1. Типовий файл конфігурації Grunt

Плагіни - це повторно використовуваний код, який визначає набір завдань. Кожен плагін внутрішньо містить каталог завдань з файлами JavaScript, які мають той же синтаксис, що і Gruntfile. Більшість плагінів Grunt публікуються з ключовим словом gruntplugin у npm та мають префікс grunt. Плагіни з'єднуються в ланцюги та зберігають тимчасовий результат в файлі. На рисунку 2.2 зображено типовий ланцюг обробки плагінами в Grunt.

Рисунок 2.2. Ланцюг обробки плагінами в Grunt.

Gulp - це інструментарій JavaScript з відкритим кодом, створений Еріком Шоффсталлом, який використовується як менеджер-задач або система потокової збірки в розробці веб-застосунків.

Менеджер задач побудований на Node.js та Npm. Він використовується для автоматизації трудомістких та повторюваних завдань, що досить часто трапляються у веб-розробці. такі як мінімізація, конкатенація, тестування компонентів та файлів, оптимізація застосунку та його ресурсів. Такими ресурсами можуть бути: медіа файли, таблиці стилів, файли розмітки та коду програми.

Для конфігурації Gulp використовує звичайний JavaScript файл, що зазвичай має назву gulpfile.js та лежить у корені проекту. Саме в цьому файлі визначається, що саме менеджер задач бути виконувати. Виконання задач покладається на плагіни, що виконують майже усі завдання.

```
var Gulp = require('Gulp');
var sass = require('Gulp-sass');

Gulp.task('sass', function () {
    return Gulp.src('./sass/**/*.scss')
        .pipe(sass().on('error', sass.logError))
        .pipe(Gulp.dest('./css'));
});

Gulp.task('sass:watch', function () {
    Gulp.watch('./sass/**/*.scss', ['sass']);
});
```

Рисунок 2.3. Зображення типової конфігурації задач Gulp

Плагін повинен добре виконувати лише те, для чого він призначений. Якщо щ плагін виконує декілька речей, його необхідно розділити на декілька плагінів. Таким чином досягається гнучкість налаштування в цілому. Екосистема gulp включає в себе майже 4000 таких плагінів[11].

Gulp - побудований на потоках NodeJs. Ці потоки полегшують з'єднання файлових операцій через трубопроводи(pipelines)[12]. Gulp зчитує файл і

передає дані з одного спеціалізованого плагіна в інший через оператор .pipe(), виконуючи одне завдання за один раз. На оригінальні файли жодним чином не впливає, поки всі плагіни не будуть оброблені. В залежності від налаштувань оригінали файлів можуть бути змінені, або ж можуть бути створені нові. Це надає можливість виконувати складні завдання, пов'язуючи численні плагіни. Користувачі також можуть писати свої власні плагіни для визначення власних завдань. [13] На відміну від інших виконавців завдань, які виконують завдання за конфігурацією, gulp вимагає знання JavaScript та програмування для визначення його завдань.

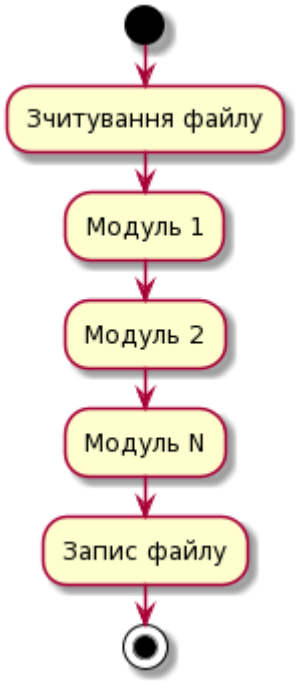


Рисунок 2.4. Схема роботи менеджера задач Gulp із файлами.

Порівняння Grunt и Gulp. На перший погляд здається, що ці менеджери задач однакові, однак це не так.

Основною відмінністю є спосіб зберігання проміжних результатів у ланцюгу обробки. Grunt використовує файли, в той час як Gulp використовує оперативну пам’ять.

Також потрібно враховувати те, що виконує Grunt завдання одне за одним, а Gulp може виконувати декілька задач одночасно. В таблиці 2.2 наведено порівняння менеджерів задач за основними характеристиками.

Таблиця 2.2

Порівняння Grunt и Gulp

	Grunt	Gulp
Формат конфігурації	JSON об’єкт	JavaScript код
Збереження тимчасових результатів	У файлах	У оперативній пам’яті
Кількість одночасних задач	Одна	Декілька
Здатний встановлювати залежності задач	Ні	Так
Формат вихідних даних	JSON	JavaScript

2.3. Збирач модулів Webpack.

Webpack не є звичайним менеджером задач JavaScript, а дещо біль спеціалізоване. Це статичний збирач модулів для сучасних програм JavaScript. [14]. Коли webpack обробляє вашу програму, він внутрішньо будує графік залежностей, який відображає кожен модуль, який потрібен вашому проекту, і генерує один або кілька пакетів. Однак, це не робить його менш значущим та функціональним.

Основною відмінністю від попередніх рішень у webpack є те, що достатньо вказати лише початкову точку входу в застосунок, а далі збирач проаналізує вміст файлу та виконає необхідні перетворення. На рисунку 2.5 зображено систему плагінів та завантажувачів, які Webpack використовує для збирання застосунків.

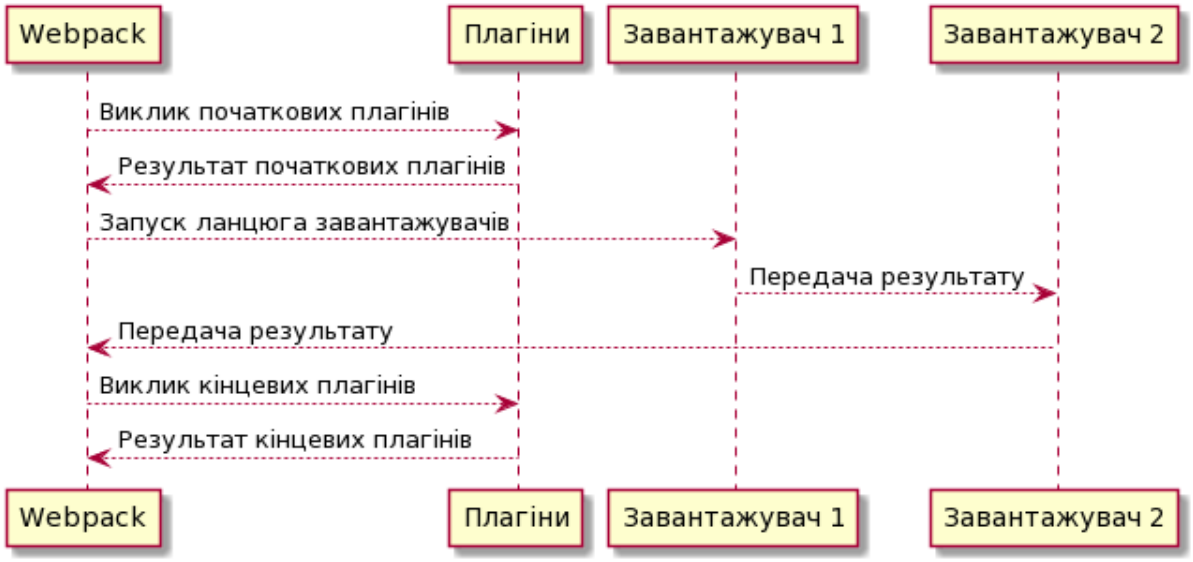


Рисунок 2.5 Схема взаємодії системи Webpack

Webpack запускається через інтерфейс командного рядка та може бути налаштований через файл конфігурації, що має назву webpack.config.js. Без додаткових засобів, Webpack може працювати лише з JavaScript файлами. Для розширення своїх можливостей збирач використовує систему завантажувачів та плагінів. Наведемо на рисунку 2.6 частину файлу конфігурації, що додає можливість Webpack працювати із файлами таблиць стилів.

```
module.exports = {
  /*...*/
  module: {
    rules: [
      {
        test: /\.css$/,
        use:
          [
            'style-loader',
            'css-loader',
          ]
      }
    ]
  }
  /*...*/
}
```

Рисунок 2.6. Частина файлу конфігурації Webpack підтримки css файлів

Завантажувачі у Webpack є аналогами завдань в Gulp. Вони приймають вміст файлів, а потім перетворять його необхідним чином і включають результат перетворення в загальну збірку. При цьому кожен завантажувач має своє певне призначення. Завантажувачі можна об’єднувати у ланцюги, якщо необхідний функціонал декількох завантажувачів.

Як продемонстровано на рисунку 2.7, перший завантажувач у ланцюгу отримує оригінальний вміст файлу. А вже після чого, кожен наступний завантажувач працює із результатом попереднього. Таким чином, можна створювати складні ланцюги перетворень із простих однонаправлених завантажувачів.

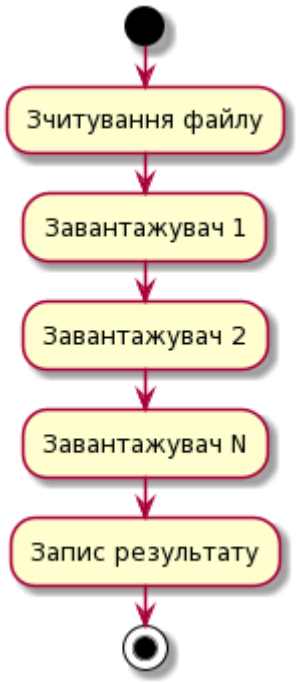


Рисунок 2.7. Схема роботи завантажувачів Webpack

Запис результату обробки файлу можливий як у файл, так і в оперативну пам’ять. Можливість збереження в оперативну пам’ять слугує зв’язком результату обробки завантажувачів та плагінів.

Підключення завантажувача відбувається за допомогою файла конфігурацій, що не є складним завданням. Потрібно додати новий об’єкт до масиву завантажувачів. Типовими полями налаштування завантажувачів є:

- test - регулярний вираз файлів, що потрібно включити в обробку завантажувача
- exclude - регулярний вираз файлів, що потрібно виключити з обробки завантажувача
- loader – безпосередньо сам завантажувач, що слід включити в лацюг завантажувачів
- options – налаштування завантажувача та його додаткова конфігурація

На рисунку 2.8 зображено приклад підключення ланцюга завантажувачів таблиць стилів. Спочатку sass-loader перетворює таблиці стилів на мові sass в мову css. Далі css-loader працює з імортами стилів та готує необхідні стилі, що будуть додані в веб-сторінку за допомогою style-loader.

```
module.exports = {
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          { loader: 'style-loader' },
          {
            loader: 'css-loader',
            options: {
              modules: true
            }
          },
          { loader: 'sass-loader' }
        ]
      }
    ]
  }
};
```

Рисунок 2.8. Ланцюг завантажувачів таблиць стилів

Плагіни Webpack виконують ті ж функції, що і завантажувачі, проте в них є більше можливостей. Вони не прив’язані до вмісту файлу, а навпаки існує можливість вибору на якому етапі компіляції плагін повинен

спрацювати. Конфігурація та параметри плагінів також більш гнучкі, аніж завантажувачів.

Однак, на відміну від завантажувачів, плагіни працює не з окремим файлом, а з цілим пакетом файлів. Плагіни можуть виконувати свої функції до або після перетворень всередині завантажувачів.

Налаштування плагінів, так як і завантажувачів, відбувається у файлі конфігурації. Плагіни встановлюються за допомогою пакетного менеджера Npm та додаються у масив плагінів. Налаштування плагіну відбувається через передачу йому об’єкту конфігурації у конструктор.

Система завантажувачів і плагінів надає повний спектр можливостей під час збірки застосунку.

Основними з них є:

- перетворення коду програм;
- аналіз коду;
- мінімізація файлів;
- конкатенація файлів;
- створення динамічних імпортів для частин програм за деревом залежностей;

Хоча й конфігурація Webpack складніше за Gulp, однак сучасні фреймворки розробки SPA такі як: React, Vue, Angular використовують саме його для збірки застосунків, що є вагомим аргументом на користь Webpack Його функціональні можливості та ефективність вищі ніж у схожих рішень, що в свою чергу, призводить до зростання його популярності.

На рисунку 2.9 зображено приклад повноцінного файлу конфігурації, що зображає принцип підключення завантажувачів, плагінів та їх конфігурації.

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const webpack = require('webpack');
const path = require('path');

module.exports = {
  entry: './index.js',
  output: {
    filename: 'main.bundle.js',
    path: path.resolve(__dirname, 'dist')
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        use: 'babel-loader'
      }
    ]
  },
  plugins: [
    new webpack.ProgressPlugin(),
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};
```

Рисунок 2.9. Повний файл конфігурації Webpack

2.4. Розробка завантажувачів

Завантажувач - це модуль NodeJs, який експортує функцію. Ця функція викликається, коли ресурс повинен бути перетворений цим завантажувачем. Дана функція матиме доступ до API завантажувача, використовуючи контекст, наданий їй.

Кожен завантажувач повинен відповідати вимогам до завантажувачів. Відповідно до офіційної документації[2-5], розміщеної на сайті Webpack, такими вимогами є:

- Простота реалізації. Кожен завантажувач виконує лише одне спеціалізоване завдання Метою такого рішення є полегшення розробки і підтримки завантажувача. Також це допомагає створювати більш гнучкі ланцюги завантажувачів.
- Підтримка ланцюгів завантажувачів. Завантажувач повинен підтримувати можливості створення ланцюг. Не потрібно

створювати модуль, що буде виконувати декілька задач, потрібно використовувати по одному модулю на одну задачу. Розділяючи їх, можна не просто зробити кожен навантажувач простим, але й використовувати їх для того, про що ви раніше не думали.

- Використовуйте модульний підхід. Модулі, створені завантажувачем, повинні дотримуватися тих же принципів проектування, що і звичайні модулі.
- Незалежність і постійність. Переконайтесь, що навантажувач не зберігає стан чи контекст між перетвореннями модулів. Результат роботи завантажувача жодним чином не залежить від попередніх чи наступних завантажувачів та не зберігає жодних даних про своє попереднє виконання.
- Використовуйте допоміжні функції завантажувачів. Як згадували вище, кожен завантажувач має доступ до API завантажувачів, що надає Webpack. Майже кожен модуль отримує параметри та повинен перевіряти їх перед запуском. Саме тому, ці функції поставляються в додаткових модулях Webpack. На рисунку 2.10 зображено їх використання.

```
import { getOptions } from 'loader-utils';
import validateOptions from 'schema-utils';

const schema = {
  type: 'object',
  properties: {
    path: {
      type: 'string'
    },
    isStrict: {
      type: 'boolean'
    }
  }
};

export default function(source) {
  const options = getOptions(this);

  validateOptions(schema, options, 'Some loader');

  // обробка вхідного файлу програми

  return result;
}
```

Рисунок 2.10. Приклад використання допоміжних функцій

- Визначте залежності завантажувача. Це стосується як додаткових модулів так і читання із файлової системи. Дана інформація необхідна Webpack для правильного функціонування кешування та роботи іншим пов’язаних завантажувачів.
- Уникайте дублювання коду. Якщо існує повторювана частина програми, винесіть її в окремий підмодуль.
- Уникайте використання абсолютних шляхів. Вони призводять до руйнування зв’язків та роблять роботу кеш алгоритму неможливою. Додаткові функції завантажувачів мають інструмент, що перетворює абсолютний шлях у відносний.
- Вказуйте прямі залежності завантажувачів. Якщо ваш завантажувач є обгорткою над іншим - додайте його як залежності в секцію peerDependencies.

Слідуючи цим правилам, можна створити корисний усім розробникам завантажувач, що буде чудово комбінуватись із іншими завантажувачами.

2.5. Опис функціональності завантажувача

Головною метою розробки завантажувача є надати можливості та інструменти розробнику для виключеня блоків коду із кінцевої програми або ж його виконання, під час збірки проекту. При цьому функціонал, не повинен залежати від типу файлу, що обробляється.

Щоб виділяти певні блоки коду, розробнику повинен бути наданий синтаксис, який розуміє завантажувач. Його функціями буде:

- виділення блоку коду
- виділення умови

За основу такого блоку буде взято спрощену конструкцію умовних виразів. Вони добре відомі усім розробникам та є базовим елементом будь-якої мови програмування. Синтаксичною одиницею, навколо якої буде побудовано модуль є коментар. Дане рішення має наступні переваги:

- відсутність синтаксичних помилок
- можливість роботи програми, у випадку відключення модуля
- сумісність із інтегрованими середовищами розробки IDE
- валідність під час аналізу коду програми

В залежності від синтаксису та умови буде прийняте рішення, про обробку блоку коду. Можливими сценаріями є:

- запис в логи під час компіляції
- виконання коду під час компіляції
- залишити або видалити блок коду із фінальної версії програми
- Запропонована конструкція складається з наступних компонентів:
- // <<< # {code} >>> - блок запису в логи
- // <<< = {code} >>> - блок виконання коду
- // <<< ?(flag) {code} >>> - умовний блок видалення коду

Після завершення роботи завантажувача, жодна з цих конструкцій не повинна залишитись у коді програми, адже свою функцію вона вже виконала.

На випадок, якщо дані конструкції не можуть бути використані повинна бути можливість їх змінити, через файли конфігурації, без зміни коду самого модуля.

2.6. Розробка алгоритму роботи модуля

Перед початком розробки модуля, необхідно попередньо розробити алгоритм робити модуля. Першим етапом є створення загального алгоритму дій модуля підготовки односторінкових застосунків до публікації.

На рисунку 2.11 наведено алгоритм роботи модуля підготовки односторінкових застосунків до публікації на абстрактному рівні.

Рисунок 2.11. Алгоритм роботи модуля

Блок схема алгоритму включає в себе два основні блоки, що потрібно реалізувати для повноцінної роботи модуля. Цими блоками є створення масиву блоків та обробка блоку коду. Спочатку розглянемо алгоритм створення масиву блоків.

Враховуючи той факт, що було обрано формат блоків не лише із початковими символами, а й кінцевими, розробка алгоритму виділення цих блоків значно спрощується. На рисунку 2.12 зображено алгоритм створення масиву блоків.

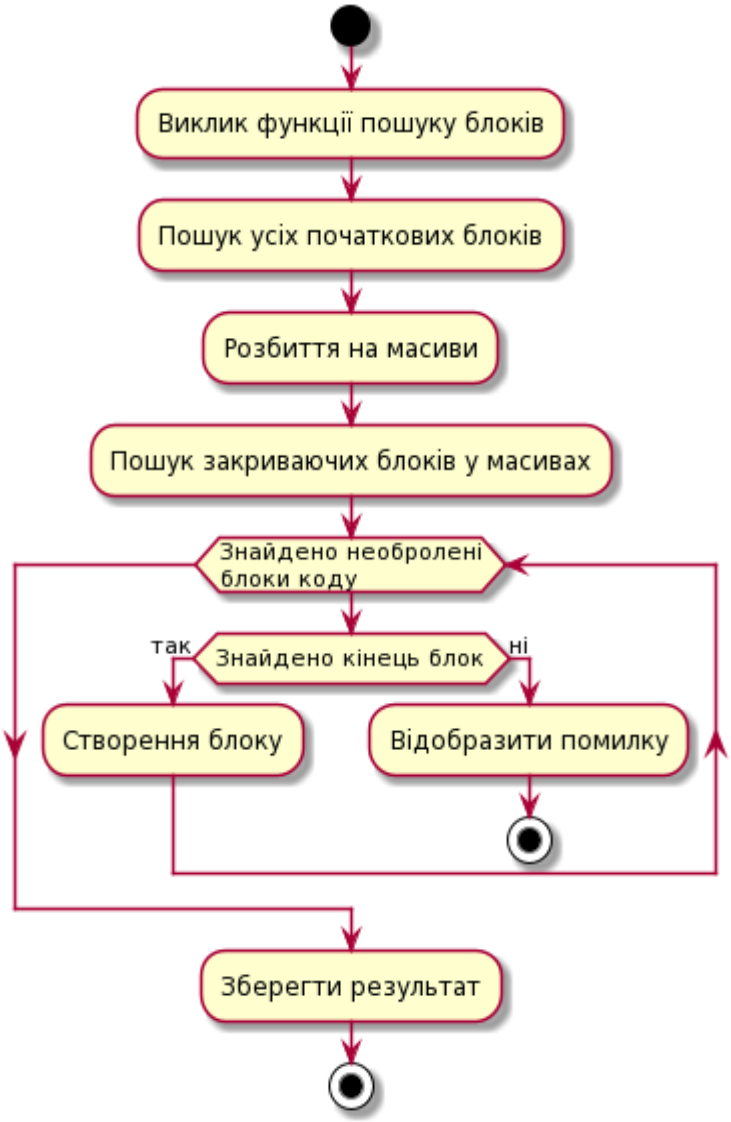


Рисунок 2.12. Алгоритм створення масиву блоків

Блоки коду, що мають різне призначення відрізняються синтаксично на початку, тому після етапу знаходження блоків вже відомо як їх обробляти. Але потрібно враховувати можливість різних дій, тому алгоритм обробки блоків коду в своїй основі несе. На рис. 2.13 зображений алгоритм обробки блоку коду.

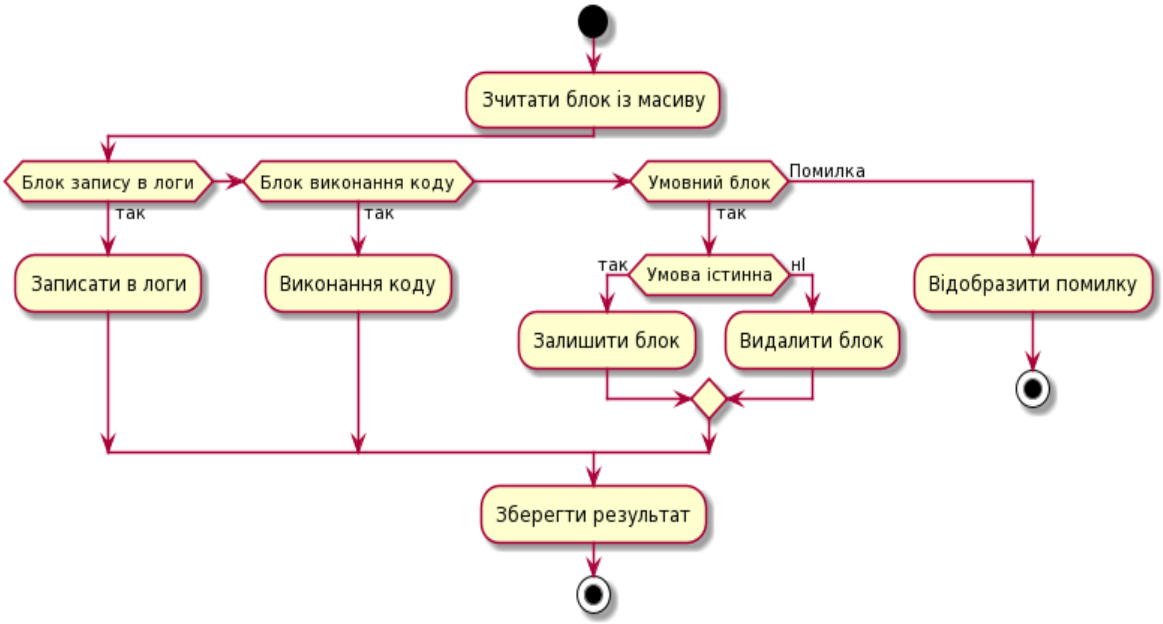


Рисунок 2.13. Алгоритм обробки коду

Сукупність усіх алгоритмів, наведених вище, повністю задовільняє функціональні потреби модуля. Їх простота є наслідком вибору зручного формату блоків, що добре сприймаються як під аналізу, так і зручні для розробників.

2.7 Взаємодія з іншими модулями

Враховуючи те, що сучасні з системі підготовки односторінкових застосунків до публікації з'єднують модулі у ланцюги, тому потрібно правильно їх з'єднати.

Вирішення цієї задачі є досить простим, однак необхідним етапом. Враховуючи той факт, що певні блоки коди можуть бути додані або видалені нашим модулем, він повинен бути першим у ланцюгу модулів. Таким чином, усі подальші перетворення будуть виконані.

На рисунку 2.14 зображено типовий ланцюг включення модулів у системах підготовки односторінкових застосунків.

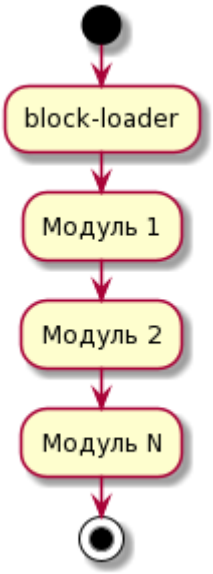


Рисунок 2.14. Послідовність включення модуля

Використання розроблювального завантажувача у ланцюгу із іншими модулями може значно покращити продуктивність роботи веб-додатку, в тому числі надає змогу створювати різні типи конфігурації та версій веб-застосунку.

ВИСНОВКИ ДО РОЗДІЛУ 2

В даному розділі було проведено загальний опис предметної області задачі. Визначено функціональності можливості системи.

Дослідженні сучасні системи збірки проектів. Найбільш гнучким та потужним інструментом виявився webpack. Проаналізовано архітектуру даної системи збірки проектів, його основні компоненти та принципи роботи. Виділено основні етапи створення завантажувачів Webpack.

Враховуючи особливості поставленої задачі та особливості інструментів Webpack було прийнято рішення, що найкращим способом реалізації є завантажувачі. Було визначено алгоритми роботи модуля та спосіб його підключення в типовий проект.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Вибір мови програмування

Враховуючи середовище використання та виконання модулю, найкращим вибором мови програмування є JavaScript або мови компільовані в нього. Цей висновок можна зробити на основі наступних факторів:

- Модуль будуть використовувати веб-розробники, тому мова JavaScript та похідні від неї добре їм відомі.
- Системою збірки проекту є Webpack і середовищем виконання є NodeJs, що розроблена спеціально для мови JavaScript.
- Завантажувачі в Webpack працюють на основі функцій і повинні бути простими, що чудово забезпечує мова програмування JavaScript

Також позитивною стороною такого рішення є те, що модуль можна бути використовувати не лише для створення односторінкових застосунків, а й для повноцінних додатків на платформі Electron або React Native. Вони використовують систему збірки Webpack, що робить модуль автоматично доступним для них.

Основними перевагами мови є:

- Простота у вивченні та реалізації.
- Одна із найпопулярніших мов програмування світі.
- Універсальність. З виходом платформ NodeJs, Electron, React Native та інших мова перестало обмежуватись виконанням у браузері та веб-додатках, а стала повноцінною мовою програмування розробки мульти-платфоременних додатків.
- Швидкість. Хоча й компіляція відбувається безпосередньо перед виконання та компілятор V8 від гугл чудово працює і забезпечує високу швидкість виконання.
- Мова швидко розвивається та з кожним роком збільшує свої можливості та перелік інструментів для розробника.

Мови що компілюються в JavaScript є також не поганим варіантом. Однак, переваги їх використання значно менші за недоліки. Основою перевагою таких мов як TypeScript та Coffee Script є розширення синтаксичних можливостей. Це стає помітно на великих проектах при використанні сотень сутностей, а при створенні модулю.

Недоліками також є ускладнення створення та підтримки модулю. Це ще одна додаткова технологія яку потрібно вивчати. Необхідно вірно налаштувати конфігурацію для їх підтримки. Також з їх використання зникає гнучкість, присутня при розробці на JavaScript.

Підсумовуючи все описане вище було прийнято рішення використовувати мову JavaScript для розробки модулю підготовки односторінкових застосунків до публікації.

3.2 Вибір фреймворку тестування

У сучасному світі помилки у розробці програмного забезпечення може коштувати дуже дорого. Компанії втрачають гроші та репутацію через це. Ось чому тестування є важливим етапом розробки програми. Враховуючи те, що розроблюваний завантажувач є початком у ланцюгу, його коректна робота є обов'язковою.

Сучасні методології автоматичного тестування програмного забезпечення виділяють наступні основні методології тестування:

- Модульне тестування - тестування програмного забезпечення, при якому перевіряються окремі підпрограми, класи або процедури в програмі. Методологія полягає у тестуванні малих блоків програми, з яких пізніше буде побудована програма.
- Інтеграційне тестування – тестування програмного забезпечення, при якому перевіряють окремі процеси або підсистеми, які складаються із декількох компонентів. Під час такого тестування перевіряється як окремі модулі та компоненти комбінуються один з одним та результат їх спільної роботи.

- Системне тестування – тестування, при якому перевіряють роботи системи в цілому, намагаючись максимально відтворити сценарії реального використання продукту.

Для перевірки роботи модуля необхідно і достатньо покрити розроблений код програми модульними тестами. При цьому, тести повинні повністю охоплювати можливі ситуації та функціональність модуля, шляхом порівняння вихідних результатів з очікуваними при різних вхідних даних.

Оскільки тестування давно є невід'ємним етапом розробки програмного забезпечення, для зручності розробників були створені фреймворки тестування, що дозволяють автоматизувати рутинні задачі та надають певні інструменти тестування розробникам.

Порівняння фреймворків тестування. Саме тому необхідно обрати зручний та функціональний фреймворк тестування враховуючи потреби та методологію тестування.

Mocha - багатофункціональна тестова система для NodeJs. Вона чудово поєднується із іншими бібліотеками, що є беззаперечною перевагою. Також перевагою є те, що система повноцінно документована. Однак, і недоліків у цієї системи вистачає. Немає вбудованих додаткових інструментів для порівняння отриманого і еталонного результату, тому бібліотеку часто використовують у сукупності із бібліотекою Chai. Ще одним недоліком є необхідність створення і налаштування конфігурації.

Chai - це бібліотека тверджень та функцій порівняння, яка чудово приємно поєднуватися з будь-яким іншим фреймворком тестування JavaScript.

Бібліотека має велику кількість плагінів, розширень та є частиною багатьох тестових систем. Вона чудово виконує свою роботу, однак, це лише бібліотека тверджень, тому досить рідко використовується самостійно.

Jasmine - це фреймворк тестування з відкритим кодом для JavaScript. Він спрямований на роботу на будь-якій платформі з підтримкою JavaScript, не залежно від додатку, IDE та із синтаксисом, який легко читається. Тому його

можна використовувати для тестування будь-яких JavaScript додатків, що є вагомим фактором.

Jasmine можна сміливо використовувати самостійно, адже він надає повний спектр інструментів та можливостей необхідних при модульному тестуванні. Однак, це не означає що функціональність фреймворку не можна розширити додатковими бібліотеками.

Jest - це фреймворк тестування JavaScript, призначений для забезпечення коректності будь-якої бази коду JavaScript. Це дозволяє писати тести за допомогою доступного, звичного та багатофункціонального API, який швидко дає результати.

Jest добре задокументований, вимагає невеликої конфігурації і може бути розширений, щоб відповідати вимогам будь-якого проекту.

Основними перевагами Jest є:

- Відсутність обов’язкової конфігурації. Фреймворк готовий до роботи одразу після встановлення. Тож розробник більше сфокусований на розробці тестів, а не конфігурації фреймворку.
- Паралельний запуск тестів, що дозволяє швидше опрацювати великі набори тестів.
- Можливість працювати самостійно та в сукупності із іншими фреймворками та бібліотеками. Jest має все необхідна для самостійної роботи одразу після встановлення.
- Повноцінна документація, що описує усі можливості фреймворку у деталях.

Отже, існує багато фреймворків із своїми перевагами та недоліками які потрібно використовувати для своїх цілей. Заради кращого розуміння в таблиці 3.1 наведено порівняння перелічених вище бібліотек та фреймворків за основними характеристиками.

Таблиця 3.1

Порівняння фреймворків тестування

	Mocha	Chai	Jasmine	Jest
Тестування користувацького інтерфейсу	-	-	+	+
Вбудовані інструменти порівняння	-	+	+	+
Можливість самостійної роботи	-	-	+	+
Відображення прогресу та результатів тестів	+	-	+	+
Підтримка заміни даних	+	-	+	+
Звіт про покриття коду тестами	-	-	-	+

Враховуючи результати порівняння, було прийнято рішення використовувати Jest для тестування модуля. Він не потребує додаткового налаштування, надає усі необхідні інструменти та можливості, має простий інтерфейс та повноцінну документацію. При цьому фреймворк не має чітко виражених недоліків.

3.3 Розробка налаштувань завантажувача

Перед початком розробки безпосередньо завантажувача, необхідно визначити можливі вхідні параметри для нього.

Налаштування модуля можна розділити на дві групи:

- Параметри модуля – це параметри, які визнають роботу модуля та використовуються функціями завантажувача.
- Параметри користувача – це параметри, які необхідні для функціонування блоків програми, що будуть оброблюватись розроблюваним завантажувачем.

Такий простий розподіл надає широкі можливості для налаштувань модуля. Параметри модуля починаються із двох символів нижнього підкреслення. Даний формат обрано з метою уникнення можливості колізій

між параметрами модуля і користувача. Формат параметрів користувача є вільним і обмежений лише функціональністю та синтаксисом мови JavaSctipt.

Параметри модуля визнають синтаксис блоків коду, але вони не є обов'язковими. Вони виділені з метою створення максимально гнучкої системи, яка не залежить від технологій проекту. Модуль має наступні параметри:

- __startBlockRegex – регулярний вираз початку блоку;
- __endBlockRegex – регулярний вираз кінця блоку;
- __evalBlock - початок блоку на виконання;
- __logBlock - початок блоку логів;
- __conditionBlock - початок умовногоблоку;
- __stopBlock – початок блоку зупинки;
- __endBlock - кінець блоків.

Приклад передачі параметрів користувача та власних завантажувачу користувацьких так і зображено на рисунку 3.1.

```
/* ... */
{
  test: /\.(js|mjs|jsx|ts|tsx)$/,
  enforce: 'pre',
  use: [
    {
      loader: require.resolve(`${paths.appSrc}/loader`),
      options: {
        mode: 'dev',
        appId: '1D22AC21187F',
        __endBlock: '=>>'
      },
    },
  ],
  include: paths.appSrc,
},
/* ... */
```

Рисунок 3.1. Передача параметрів завантажувачу

3.4 Розробка завантажувача

Розробка завантажувача розпочинається із написання головної функції, яка буде експортована в подальшому. На рисунку 3.2 зображено початок цієї функції.

```
function loader(source, options = {}) {
  options = getOptions(this) || options;

  // Webpack cache function
  if (this.cacheable) {
    this.cacheable();
  }

  const startBlockRegex =
    options.__startBlockRegex || /\/\/\s?(<<<[=,#,?]{0,1})/;
  const endBlockRegex = options.__endBlockRegex || /\s?(>>>)/;

  const isEvalBlock = (p) => p === (options.__evalBlock || '<<<=');
  const isLogBlock = (p) => p === (options.__logBlock || '<<<#');
  const isConditionBlock = (p) => p === (options.__conditionBlock || '<<<?');
  const isStopBlock = (p) => p === (options.__stopBlock || '<<<');
  const isEndBlock = (p) => p == (options.__endBlock || '>>>');

  if (!startBlockRegex.test(source)) {
    // Blocks not found
    return source;
  }
```

Рисунок 3.2 Початок головної функції завантажувача

Функція приймає два параметри:

- source – код файлу, що в даний момент оброблюється.
- options – як параметр використовується лише при тестування. У реальному застосунку в подальшому замінюється на об'єкт options із файлу конфігурації.

Далі за потреби викликається функція кешування Webpack, її виклик необхідний для коректної роботи системи збірки. Після чого визнаються синтаксичні конструкції для пошуку у файлі, ними є передані конструкції через файл конфігурації або конструкції за замовчування. Завершується даний фрагмент перевіркою, чи є у файлі шукані синтаксичні конструкції і якщо їх немає повертається код файлу без змін.

На рисунку 3.3 зображено продовження головної функції завантажувача. За допомогою виклику допоміжних функції parse виділяються окремі токени,

з яких в подальшому будуть виділені синтаксичні блоки. Функція findAndEvalBlocks перебирає масив токенів та визнач синтаксичні блоки, які одразу і оброблює. У кінці повертається оброблений вміст файлу вже без спеціальних синтаксичних конструкцій або помилка, що передається

Webpack за допомогою виклику метода this.callback.

```
try {
  const parseParams = { source, startBlockRegex, endBlockRegex };
  let srcBlocks = parse(parseParams);

  const findAndEvalParams = {
    srcBlocks,
    options,
    isEvalBlock,
    isLogBlock,
    isConditionBlock,
    isStopBlock,
    isEndBlock,
  };
  let result = findAndEvalBlocks(findAndEvalParams);

  return result;
} catch (e) {
  this.callback(e);
}
```

Рисунок 3.3 Продовження головної функції завантажувача

Розробка функції parse. Метою і результатом функції parse є масив токенів, для подальшої обробки. В основі її роботи лежить розбиття вхідного тексту за допомогою регулярних виразів.

Регулярний вираз у мові JavaScript є об’єктом. Він може бути побудований з конструктором RegExp або записаний за допомогою літералів, додаючи у шаблон символи косої риски вперед /.

Регулярні вирази є і жахливо незручними, і надзвичайно корисними. Їх синтаксис дуже складно зрозуміти людині, а інтерфейс програмування, який JavaScript забезпечує для них не вирішує цієї проблеми. Але вони є потужним інструментом для огляду та обробки рядків. Правильне розуміння регулярних виразів робить дозволяє виконувати майже будь-які операції з рядками не витрачаючи багато часу на написання парсера.

Саме тому технологію було використання для розбиття коду файлу на масив токенів. За допомогою ланцюгу стандартних методів масиву JavaScript

та використання регулярних виразів було виконано розбиття. На рисунку 3.4 зображена функція розбиття на масив токенів.

```
function parse({ source, startBlockRegex, endBlockRegex }) {
  let srcBlocks = [];

  source
    .split(startBlockRegex)
    .map((chunk) => {
      return chunk.split(endBlockRegex);
    })
    .forEach((chunk) => {
      srcBlocks = srcBlocks.concat(chunk);
    });

  return srcBlocks;
}
```

Рисунок 3.4 Функція створення масиву токенів

Розробка сутності Blocks. Враховуючи те, що оброблюється декілька видів блоків, з'явилась необхідність відрізняти їх всередині програми. Для такої цілі чудово б підійшли перерахування.

Перерахування - це визначений користувачем тип, що складається з набору іменованих констант. Використання перерахувань збільшує рівень абстракції та дозволяє програмісту концентруватись на значення, а не перейматися тим, як вони зберігаються та як отримати доступ до них. Це зменшує ймовірність виникнення помилок.

Однак, на превеликий жаль, мова JavaScript не має вбудованих перерахувань. Тому для вирішення цієї проблеми використовується додаткова сутність Blocks. Технічно, це звичайний об'єкт із полями, що відповідають заданим значенням. Проте такий підхід дозволяє визначити типи в одному місці та використовувати у будь-якому місці програми, що нам і потрібно. На рисунку 3.5 зображена сутність Blocks.

```
const BLOCKS = {
  log: 'log',
  eval: 'eval',
  block: 'block',
  stop: 'stop',
};
```

Рисунок 3.5. Сутність Blocks

Розробка функції findAndEvalBlocks. Наступним етапом роботи завантажувача є виділення блоків та їх обробка. Цим займається функція findAndEvalBlocks. Вона приймає масив токенів як параметр, визначає що це за блок та викликає функцію обробки блоку. На рисунку 3.6 зображена частина даної функції, що працює за алгоритмом розробленим у другому розділі.

```
srcBlocks.forEach((chunk, index) => {
  if (isLogBlock(chunk)) {
    flag = BLOCKS.log;
    return;
  } else if (isEvalBlock(chunk)) {
    flag = BLOCKS.eval;
    return;
  } else if (isConditionBlock(chunk)) {
    flag = BLOCKS.block;
    return;
  } else if (isStopBlock(chunk)) {
    flag = BLOCKS.stop;
    return;
  } else if (isEndBlock(chunk)) {
    /* call eval block */
    return;
  }

  if (flag) {
    buffer.push(chunk);
  } else {
    if (prevFlag === BLOCKS.block) {
      buffer.push(chunk);
    } else {
      result.push(chunk);
    }
  }
}
```

Рисунок 3.6. Визначення типу блоку

Для визначення типу блоку функція використовує допоміжні функції, які були визначені у функції parse. Для збереження інформації про тип блоку використовується сутність Blocks. У масив buffer додаються фрагменти програми, що потребують обробки. В свою чергу, у масив result додаються блоки, що залишаються без змін.

Розробка функції evalBlock. Останнім етапом роботи завантажувача є обробка блоків. Враховуючи те, що умови або код для виконання є валідним кодом JavaScript було прийнято рішення використовувати функцію eval для запису в логи, перевірки умов та виконання блоків коду.

З документації по JavaScript відомо, що функція eval приймає один аргумент, який є рядком. Даний рядок представляє вираження, висловлювання чи послідовність JavaScript інструкцій. Вираз може включати змінні та властивості існуючих об'єкті. Тобто може використовувати змінні доступні у поточному контексті виконання. Функція повертає значення, отримане після виконання рядку, що передано як параметр.

Функцію не рекомендується використовувати в застосунках, так як вона може стати вразливим місцем для атак та ін’єкцій злоякісного коду. Однак, модуль працює лише під час збірки проекту, тому користувач немає доступу до цього коду і отримає лише результат його виконання.

Отже, функція чудово підходить для використання у нашому випадку.

На рисунку 3.7 зображена функція обробки блоків, розроблена за алгоритмом представленим у другому розділі.

```
function evalBlock({ options, flag, prevFlag, buffer, result }) {
  if (flag === BLOCKS.eval) {
    result.push(eval(buffer.join('')));
    buffer = [];
    flag = null;
    prevFlag = BLOCKS.eval;
  } else if (flag === BLOCKS.log) {
    eval(buffer.join(''));
    buffer = [];
    flag = null;
    prevFlag = BLOCKS.log;
  } else if (flag === BLOCKS.block) {
    flag = null;
    prevFlag = BLOCKS.block;
  } else if (flag === BLOCKS.stop && prevFlag === BLOCKS.block) {
    if (eval(buffer[0])) {
      result.push(buffer.slice(1).join(''));
    }
    buffer = [];
    flag = null;
    prevFlag = BLOCKS.stop;
  } else {
    throw new Error('END BLOCK WITHOUT START BLOCK FOUND');
  }
}
```

Рисунок 3.7. Функція обробки блоків

Розберемо принципи роботи даної функції більш детально. Функція має наступні параметри:

- options – параметр не використовується функцією безпосередньо. Але до нього є доступ із блоків запису в логи, блоків на виконання та умовних блоків, для цих цілей він і передається у функцію.
- flag – тип блоку, що оброблюється в даний момент.
- prevFlag – попередній блок.
- buffer – допоміжний масив із кодом, що потрібно обробити або додати до вихідного коду.
- result – вихідний код у форматі масиву.

Розглянемо дії для обробки блоків в залежності від типу блоку. Отже, алгоритм дій залежить від типу блоку:

- Блок виконання. Конструкція яку потрібно виконати знаходиться в масиві буферу. Потрібно перетворити масив конструкцій у рядок і обробити за допомогою функції eval. Результат виконання функції додається у результуючий масив.
- Блок логів. Функція запису логу зберігається у буферному масиві. Враховуючи, що дана функція є валідним JavaSctipt, її виконання також перекладається на функцію eval. При цьому результуючий масив не змінюється.
- Умовний блок. Обробка умовного блоку ділиться на дві частини. Спочатку фіксується початок умовного блоку і його умова. І лише коли зустрічається кінець блоку, починається обробка. Умова знаходиться в буферному масиві на першій позиції, решта елементів це блок коду який потрібно залишити або видалити із результуючого масиву.
- Помилка обробки. Якщо викликана функція обробки блоку, але не відомий тип блоку відобразити помилку.

3.5 Тестування завантажувача

Для тестування завантажувача було обрано фреймворк Jest, тому додаткові налаштування проекту не потребуються, достатньо створити файл із розширення test.js і фреймворк автоматично буде відслідковувати цей файл. Тому одразу перейдемо до розробки тестів.

Для тестування завантажувача був розроблений спеціальний підхід, алгоритм якого зображений на рисунку 3.8.

Рисунок 3.8. Алгоритм тестування завантажувача

Функції beforeEach та afterEach це спеціальні функції фреймворку Jest, які викликаються безпосередньо до та після кожного тесту відповідно. Їх використовують для підготовки параметрів, сутностей до тестування та прибирання побічних ефектів тестів. У нашому випадку вони застосовуються для тестування запису в лог, а якщо точніше заміни стандартної функції виводу на екран, на функцію виводу у масив.

З метою спрощення розробки та пришвидшення виконання тестів, було прийнято рішення не використовувати напряму файлову систему. Параметри, вхідні данні та очікуваний результат зберігаються у окремому файлу для

кожного тесту. Даний файл імпортується автоматично використовуючи стандартний імпорт файлів у NodeJs.

Кожен окремий тест призначений для перевірки певної частини завантажувача. На рисунку 3.9 зображено файл із параметрами, вхідними та вихідними даними для перевірки роботи коректності обробки блоку виконання завантажувачем.

```
export const options = {
  items: [
    { id: 1, name: 'First' },
    { id: 2, name: 'Second' },
    { id: 3, name: 'Third' },
  ],
};

export const initial = `
  // <<<=
  (() => {
    let html = '';
    for (let item of options.items) {
      html += '<li>' + item.name + '</li>';
    }
    return html;
  })();
  // >>>
  const a = 1;
`;

export const expected = `
  <li>First</li><li>Second</li><li>Third</li>
  const a = 1;
`;
```

Рисунок 3.9. Файл опцій, вхідних та вихідних параметрів для тестування

Тести, реалізовані за алгоритмом, зображеним на рисунку 3.8, є гарно структурованими і надають можливість перевірити роботу завантажувача із різними прикладами вхідних даних та параметрів.

За даним алгоритмом тестування було розроблено тести для перевірки обробки усіх можливостей завантажувача. На рисунку 3.10 зображено файл тестування, який повністю реалізує алгоритм тестування описаним вище. Тести написані за модульний принципом, розглянутим у підрозділі 3.2.

```
import loader from '../loader';

import * as evalOpt from './utils/eval';
import * as blockOpt from './utils/block';
import * as logOpt from './utils/log';

describe('Loader works in difirent cases', () => {
  let consoleOutput = [];
  let originalConsoleLog = console.log;
  beforeEach(() => {
    console.log = (...params) => consoleOutput.push(...params);
    consoleOutput = [];
  });

  afterEach(() => (console.log = originalConsoleLog));

  it('Eval block works', () => {
    const result = loader(evalOpt.initial, evalOpt.options);
    expect(evalOpt.expected).toBe(result);
  });

  it('Conditional block works', () => {
    const result = loader(blockOpt.initial, blockOpt.options);
    expect(blockOpt.expected).toBe(result);
  });

  it('Logs block works', () => {
    const result = loader(logOpt.initial, logOpt.options);
    expect(logOpt.expected).toBe(result);
    expect(logOpt.options.mode).toBe(consoleOutput[0]);
  });
});
```

Рисунок 3.10. Модульне тестування завантажувача

На рисунку 3.11 зображено звіт фреймворку Jest після виконання усіх знайдених тестів.

```
 PASS  src/tests/loader.test.js
  Loader works in difirent cases
    √ Eval block works (10ms)
    √ Conditional block works (1ms)
    √ Logs block works (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        5.929s
Ran all test suites.
```

Рисунок 3.11. Звіт фреймворку Jest

3.6 Перевірка роботи у проекті

Для перевірки роботи завантажувача було розроблено простий тестовий проект з допомогою бібліотеки React. Використовуючи консольну утиліту create-react-app було згенеровано базову структуру проекту. Дану структуру зображено на рисунку 3.12.

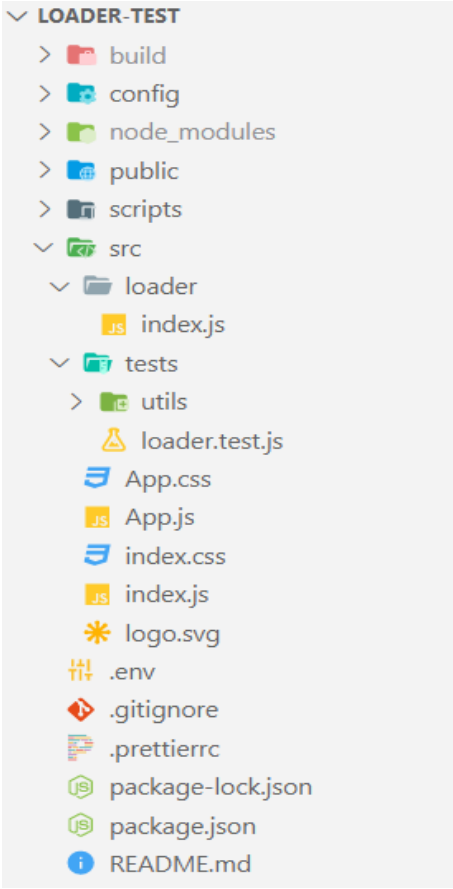


Рисунок 3.12. Структура тестового проекту

У кожної директорії є власне призначення:

- build – результат збірки проекту;
- config – файли конфігурації;
- node_modules – залежності, встановлені через Npm;
- public – додаткові статичні файли: фото, розмітка, тощо;
- scripts – згенеровані скрипти налаштування та збірки проекту;
- src – файли програми;
- tests – файли тестування та допоміжних для нього функцій;
- loader – розроблений завантажувач.

Додатково було встановлено наступні популярні пакети, redux, react-redux, redux-form, moment та jQuery. Останній необхідний лише під час розробки, що чудово демонструє можливе використання завантажувача. Залежності проекту зображені на рисунку 3.13, що є фрагментом файлу package.json. У секції scripts наведено команди збірки та запуску тестування проекту.

```
"name": "loader",
"version": "0.1.0",
"private": true,
"dependencies": {
  "@babel/core": "7.5.5",
  "@svgr/webpack": "4.3.2",
  "babel-eslint": "10.0.2",
  "babel-loader": "8.0.6",
  "babel-preset-react-app": "^9.0.1",
  "css-loader": "2.1.1",
  "dotenv": "6.2.0",
  "eslint": "^6.1.0",
  "html-webpack-plugin": "4.0.0-beta.5",
  "jest": "24.8.0",
  "jquery": "^3.5.1",
  "moment": "^2.26.0",
  "react": "^16.9.0",
  "react-dom": "^16.9.0",
  "react-redux": "^7.2.0",
  "redux": "^4.0.5",
  "redux-form": "^8.3.5",
  "style-loader": "1.0.0",
  "webpack": "4.39.1",
},
"scripts": {
  "start": "node scripts/start.js",
  "build": "node scripts/build.js",
  "test": "node scripts/test.js",
},
/* ... */
```

Рисунок 3.13. Залежності тестового проекту

Отже для початку підключимо наш завантажувач, за методологією описаною у другому розділі, а саме перед початком роботи інших завантажувачів. На рисунку 3.14 зображено підключення завантажувача перед перевіркою синтаксису файлів.

```
{
  test: /\.(js|mjs|jsx|ts|tsx)$/,
  enforce: 'pre',
  use: [
    {
      options: {
        formatter: require.resolve('react-dev-utils/eslintFormatter'),
        eslintPath: require.resolve('eslint'),
        resolvePluginsRelativeTo: __dirname,
      },
      loader: require.resolve('eslint-loader'),
    },
    {
      loader: require.resolve(`${paths.appSrc}/loader`),
      options: {
        mode: 'dev',
      },
    },
  ],
  include: paths.appSrc,
},
```

Рисунок 3.14. Підключення завантажувача у проект

Наступним етапом є підключення бібліотеки jQuery з використанням умовного блоку завантажувача. Отже, якщо параметр mode дорівнює рядку dev, тоді необхідно залишити бібліотека, якщо ні – видаляти. На рисунку 3.15 зображено, як це виглядає у проекті із використання розробленого синтаксису завантажувача.

```
import moment from 'moment';
// <<<? options.mode === 'dev' >>>
import jquery from 'jquery';
// <<< >>>
```

Рисунок 3.15. Підключення бібліотеки всередині умовного блоку

Необхідні приготування завершені, тому можемо перейти до перевірки розміру проекту. Після збірки проекту webpack відображає розмір файлів, що були створені під час збірки. Використаємо це число для перевірки розміру. Розмір файлів вказується після мініфікації та сжимання у формат gzip.

На рисунку 3.16 зображено результати збірки проекту без використання завантажувача.

```
File sizes after gzip:

  108.02 KB  build\static\js\2.4a573075.chunk.js
  777 B      build\static\js\runtime~main.3540e0a9.js
  536 B      build\static\js\main.0668dd45.chunk.js
  517 B      build\static\css\main.0c9fb83f.chunk.css
```

Рисунок 3.16. Збірка проекту без використання завантажувача

На рисунку 3.17 зображено результати збірки проекту із використання завантажувача. Webpack автоматично розраховує зміну розміру файлів між різними збірками.

```
File sizes after gzip:

  78.45 KB (-29.58 KB)  build\static\js\2.ea454974.chunk.js
  777 B                 build\static\js\runtime~main.3540e0a9.js
  524 B (-12 B)         build\static\js\main.8f8be88a.chunk.js
  517 B                 build\static\css\main.0c9fb83f.chunk.css
```

Рисунок 3.17. Збірка проекту із використанням завантажувача

Отже, дуже легко помітити, що декілька рядків умовного блоку та використання завантажувача зменшують розмір головного файлу проекту на 29.58кб., що у відсотках становить 27 відсотків розміру файлу. Даний приклад чудово ілюструє, те що модуль працює і повністю виконує свої функції.

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було проведено дослідження і аналіз технологій необхідних для розробки і тестування модуля підготовки односторінкових застосунків до публікації.

Було обрану мову програмування яка забезпечила найбільшу кількість переваг при розробці, а саме: швидкодія, універсальність та простота.

Проаналізовано сучасні підходи до тестування програмного забезпечення та обрано методологію тестування, що забезпечує перевірку роботи завантажувача. Проведено порівняння фреймворків тестування JavaScript додатків. Найкращим з яких для тестування модуля став Jest.

Розроблено зручний інтерфейс конфігурації модуля через файл конфігурації Webpack з можливістю повної адаптації синтаксису, що прибирає обмеження використання завантажувача від типу файлу та синтаксису у ньому.

Розроблено програмний код завантажувача та реалізовано алгоритми аналізу коду та обробки блоків коду. Такими блоками є: блок логів, блок виконання та умовний блок.

Створено спеціальний алгоритм тестування коректності роботи завантажувача, що забезпечує зручне автоматичне тестування модуля із різними вхідними даними.

Створено тестовий проект на основі бібліотеки React. Встановлено декілька популярних бібліотек, для максимальної схожості тестового проекту із реальними застосунками. Проведено тестування завантажувача, за результатами якого було зменшено розмір проекту на 27%, що доводить доцільність та коректність роботи розробленого модуля.

ВИСНОВКИ

Представлений дипломний проєкт присвячений розробці модуля підготовки односторінкових застосунків до публікації. Головною метою даного модуля є зменшення розміру програмного коду в кінцевому програмному забезпеченні, для пришвидшення завантаження та обробки коду.

В ході виконання проєкту було розглянуто інструменти та засоби підготовки односторінкових застосунків до публікації. Розглянуто загальні відомості, проблематику та історію розвитку модулів підготовки. Проаналізовано актуальність та необхідність розробки модуля.

Описано предметну область, вивчено безпосередньо пов’язані технології із розробкою модуля та застосунків під час збірки яких він буде використаний. Визначено основні вимоги до функціональності застосунку. Проведено детальний аналіз необхідних технологій враховуючи переваги та недоліки кожної. Розроблено алгоритми роботи модуля підготовки односторіноквих застосунків для публікації. Враховуючи визначені вимоги обрано наступні технології, платформи та фреймворки:

- Менеджер пакетів Npm
- Система збірки проекту Webpack на NodeJs
- Мова програмування JavaScript
- Фреймворк тестування Jest

Розроблено завантажувач для системи збірки Webpack, що розрізняє та оброблює три види блоків: блок логів, блок виконання та умовний блок. Спроектовано гнучкий механізм параметрів, що дозволяє у повній мірі визначати синтаксис інструкцій та блоків.

Розроблені тести для автоматичної перевірки роботи застосунку із різними наборами вхідних даних. Також створено тестовий проект для перевірки доцільності та коректності роботи модуля. За результатами збірки тестового проекту було досягнуто зменшення об’єму проекту на 27%, що є чудовим показником та підтверджує доцільність використання модуля.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

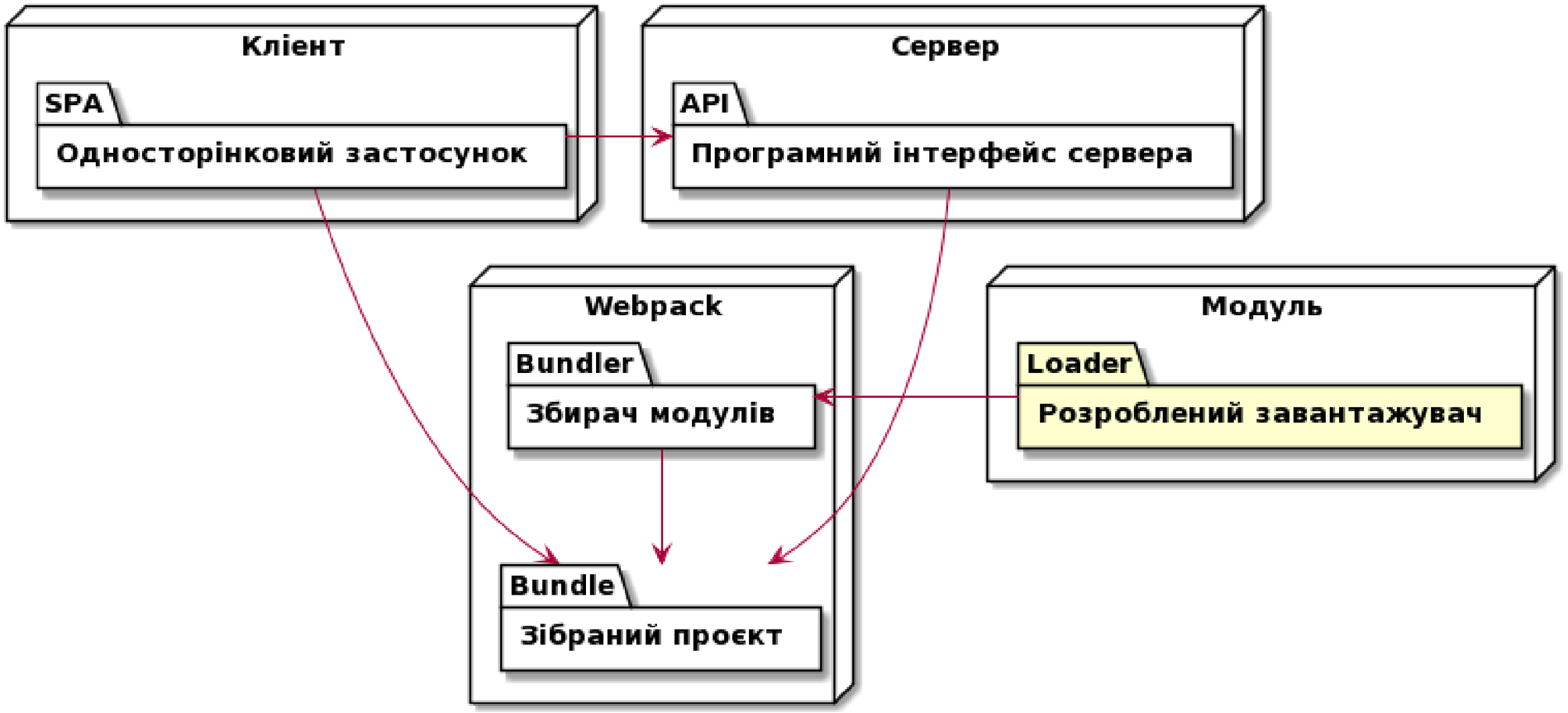
1. Kaya Ismail. What Is a Single Page Application? [Електронний ресурс] / Kaya Ismail – Режим доступу до ресурсу: https://www.cmswire.com/digital-experience/what-is-a-single-page-application/.
2. Flanagan D. JavaScript - The Definitive Guide / David Flanagan., 2006. – (5).
3. React або Angular або Vue.js — що обрати? [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/476312/.
4. Single-page application vs. multiple-page application [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.
5. How to Build A Single Page Application [Електронний ресурс] – Режим доступу до ресурсу: https://clockwise.software/blog/single-page-applications-are-they-a-good-choice-for-your-project/.
6. Crockford D. JavaScript: The Good Parts / Douglas Crockford.. – (1).
7. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming / Marjin Haverbeke.. – (3).
8. What exactly is Node.js? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/.
9. Introduction to NodeJs [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/introduction-to-nodejs.
10. JACE і як він працює [Електронний ресурс] – Режим доступу до ресурсу: http://dj-next.herokuapp.com/design/About%20JACE#/.
11. Plugins gulp.js [Електронний ресурс] – Режим доступу до ресурсу: https://gulpjs.com/plugins/.
12. Stream handbook [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/substack/stream-handbook.

13. How To Build And Develop Websites With Gulp [Електронний ресурс] – Режим доступу до ресурсу: https://www.smashingmagazine.com/2014/06/building-with-gulp/.

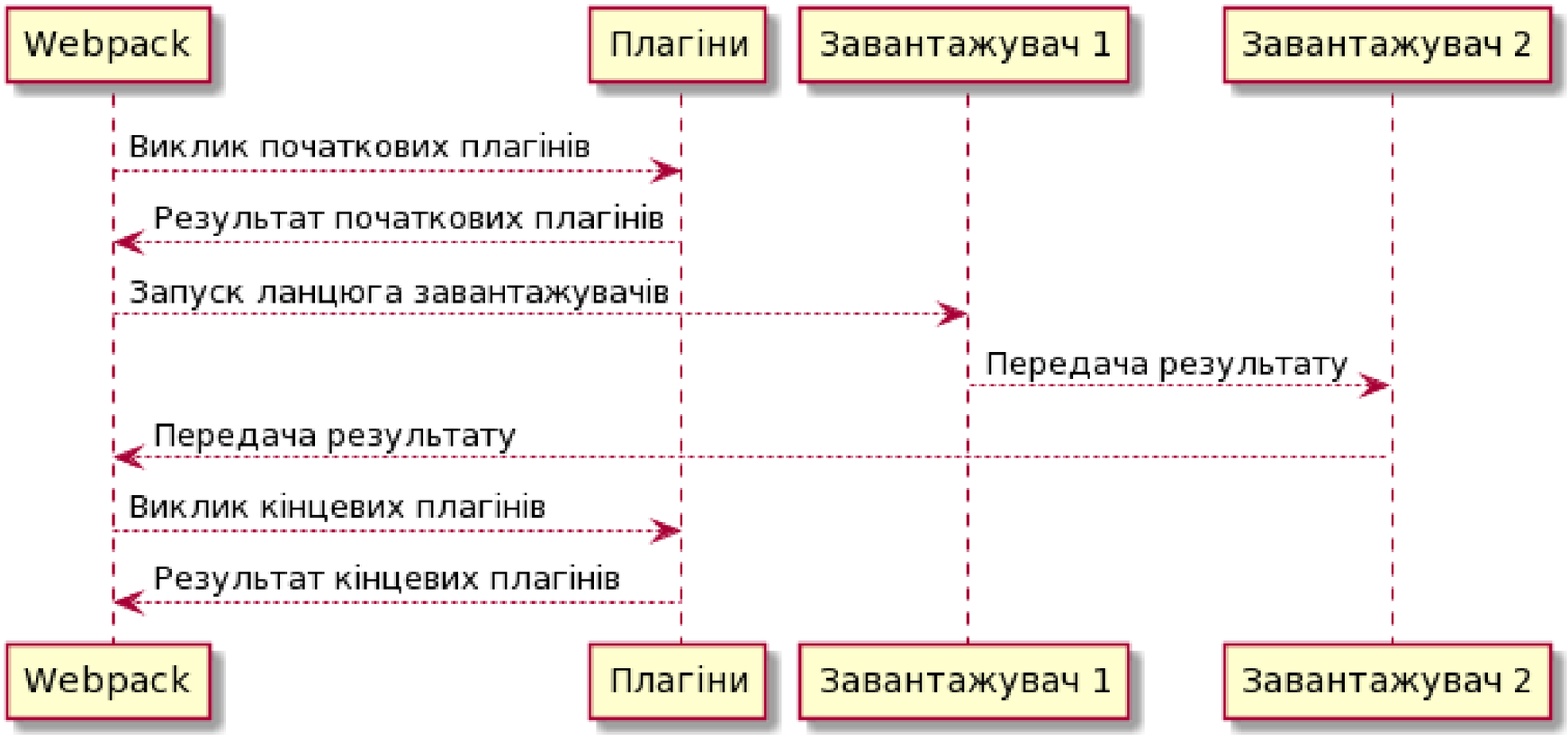
14. Webpack concepts [Електронний ресурс] – Режим доступу до ресурсу: https://webpack.js.org/concepts/.

15. Writing a Loader [Електронний ресурс] – Режим доступу до ресурсу: https://webpack.js.org/contribute/writing-a-loader/.

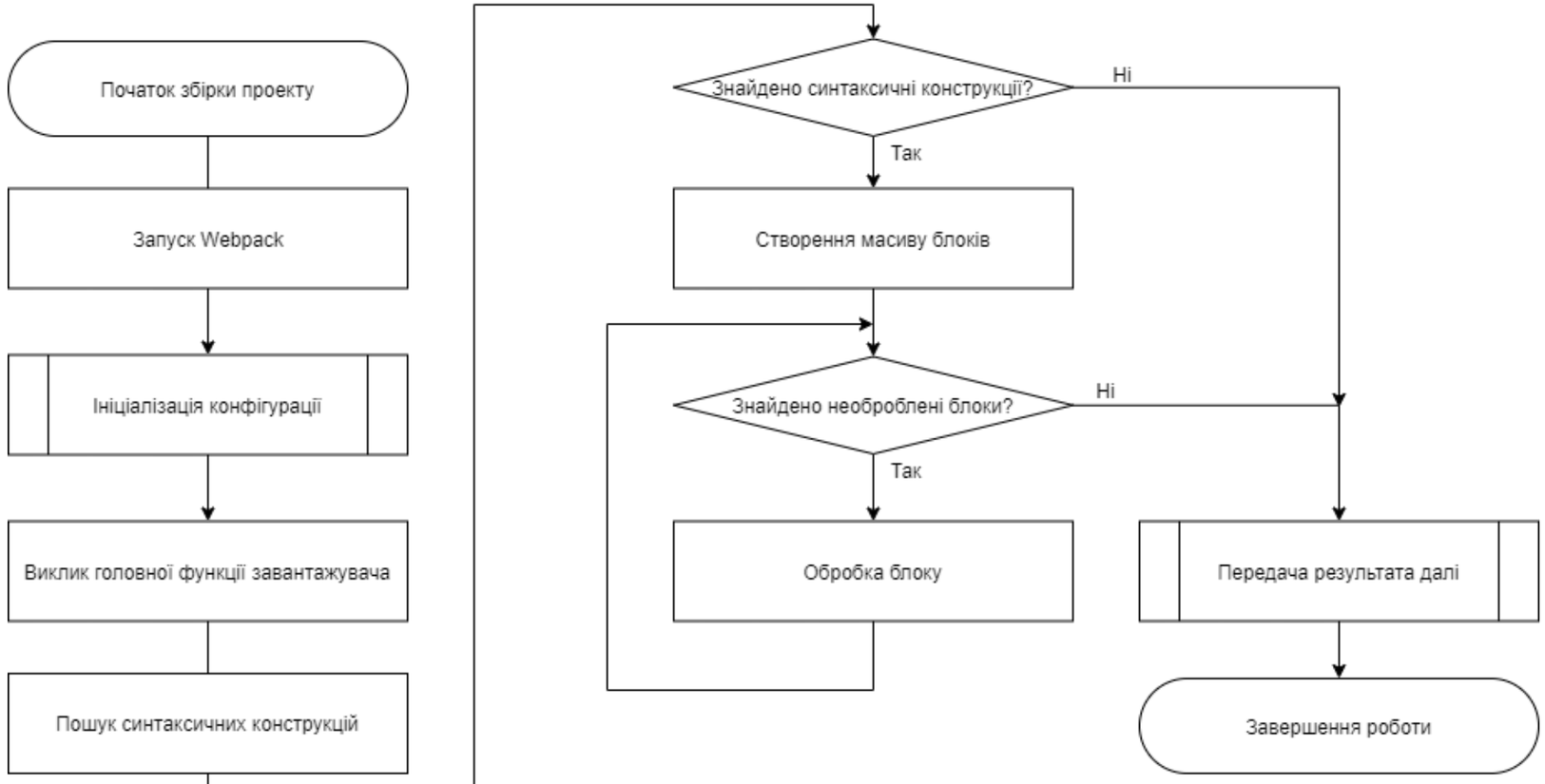
ДОДАТОК А



					ДП 6323 03.000 Д1	
					Клієнт серверна взаємодія із проектом Схема структурна	Літера / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Панасюк О.М.				Аркуш 1 / Аркушів 1
Перевір.		Болдак А.О.				
Т. контр.						НТУУ “КПІ” ФІОТ Група ІО-63
Н. контр.		Сімоненко В.П.				
Затв.						



					ДП 6323 04.000 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата	Взаємодія та передача даних між компонентами Webpack Схема функціональна	Літера / Маса / Масштаб
Розроб.		Панасюк О.М.				
Перевір.		Болдак А.О.				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоненко В.П.				НТУУ "КПІ" ФІОТ Група ІО-63
Затв.						



					ДП 6323 05.000 ДЗ	
					Алгоритм роботи завантажувача Схема принципова	Літера: — Маса: — Масштаб: —
Зм.	Арк.	№ докум.	Підпис	Дата		Аркуш 1 / Аркушів 1
Розроб.		Панасюк О.М.				НТУУ "КПІ" ФІОТ Група ІО-63
Перевір.		Болдак А.О.				
Т. контр.						
Н. контр.		Сімоненко В.П.				
Затв.						